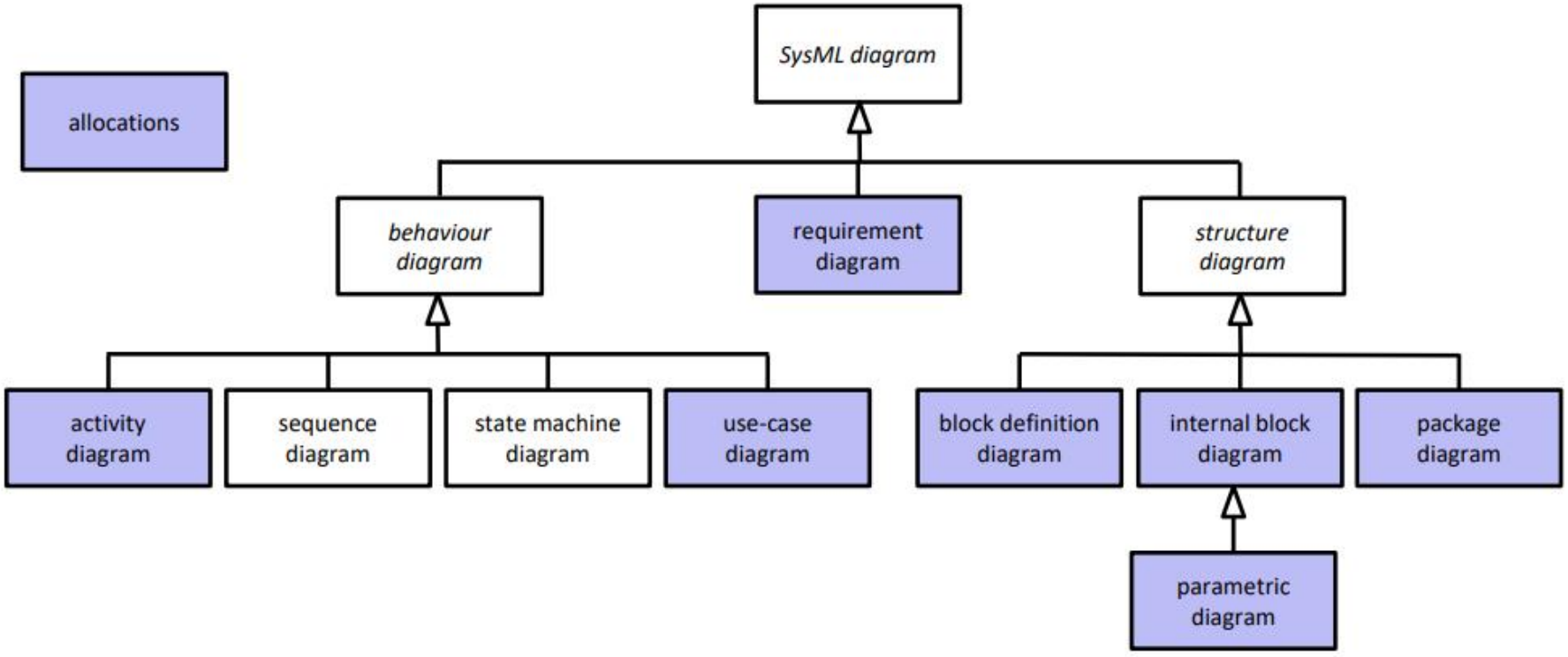
USE CASES

SysML – diagram overview

diagrams are **views** on the model
(i.e., on a subset of model elements)



SysML – diagram header

- **diagram kind:** act, bdd, ibd, pkg, par, uc, req, sd, stm
- **model-element type** (for a given kind)
  - **act:** activity
  - **bdd:** block, constraint block, package, model, model library
  - **ibd:** block
  - **pkg:** package, model, model library, profile, view
  - **par:** activity, block, constraint block
  - **req:** package, model, model library, requirement
  - **uc:** package, model, model library
  - **seq:** interaction
  - **stm:** state machine

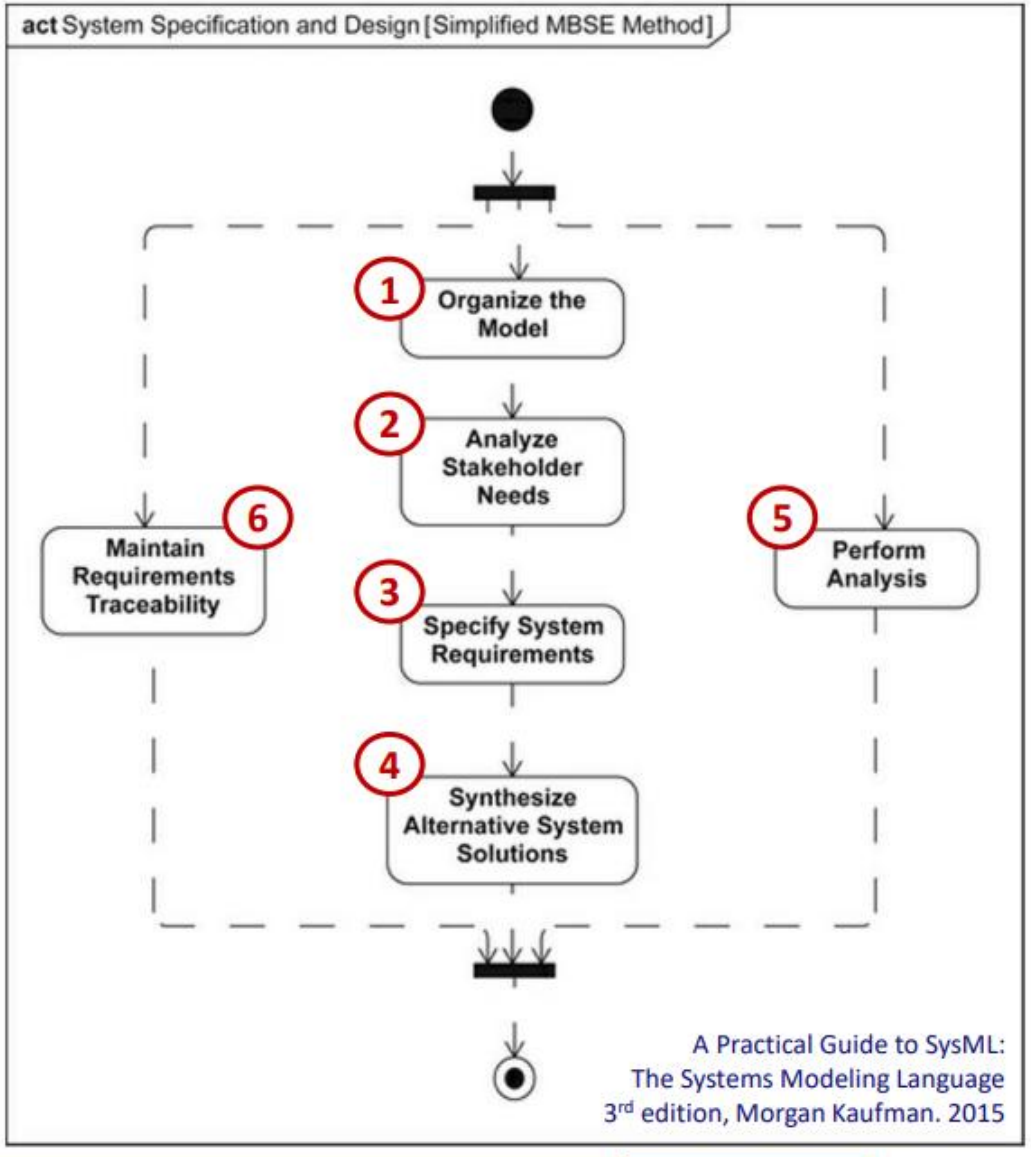
kind [type] model-element name [diagram name]

We will practice this on the go

suggested reading: section 5.2

a simplified[2] MBSE method

1. SysML package diagram
2. **stakeholders**
   SysML UC diagrams, UC descriptions
   measures of effectiveness (moes)
3. SysML requirement diagrams
4. create multiple alternatives
   - SysML BDDs – system decomposition
   - SysML IBDs – interconnections
   - SysML Activity diagrams – UC refinements
   - SysML Allocations – activities to blocks
5. - SysML PAR diagrams – covering all moes
   - POOSL models – makespan
   - verification
6. - SysML allocation – reqs to blocks/activities



SysML – use cases – model elements

model elements

- subject: system under consideration (a block; use cases are inside)
- use case: functionality of a system in terms of how it is used by its users
- actor: users of a system (e.g., human operators, subsystems in the environment, …)

actor relationships

- generalization : a specialized actor participates in all use cases of general actor

use case relationships

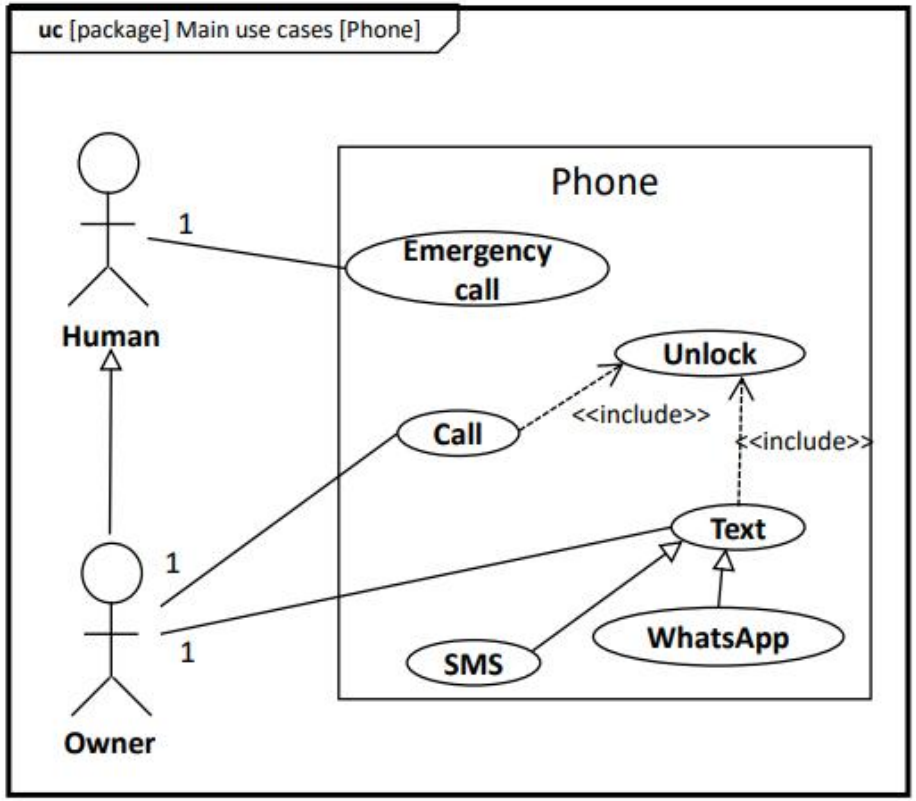
- generalization : specialized use case is involved with the same actors/scenarios as the general use case
- inclusion: the included use case is always performed as part of the base use case

relation between actors and use cases

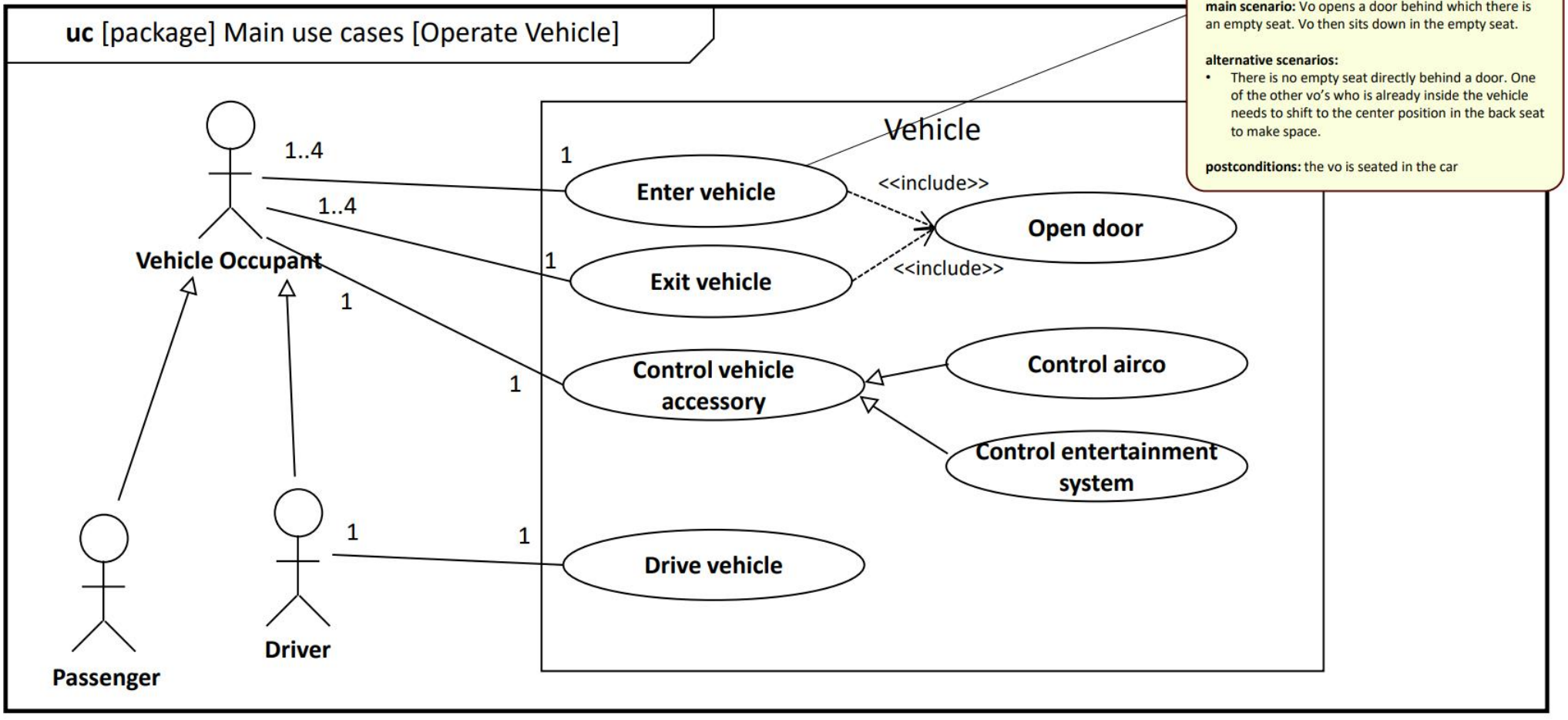
- related to use cases by bidirectional communication paths (associations)
- multiplicities on both ends, e.g., 1..* (default is 0..1)

multiplicity at actor end: numbers of actors involved in the use case

multiplicity at use-case end: number of instances in which the actor(s) can be involved at any one time



SysML – use case diagram (uc)



SysML – use cases – descriptions

the use case diagram by itself gives very little information; it must be accompanied by a use-case description that includes:

- **name**
- **primary actor**: who triggers the use case
- **supporting / secondary actors**: other participating actors
- **preconditions**: the conditions that must hold such that the use case can begin
- **main scenario**: what happens in this use case
- **alternative scenarios**:
  - list of related use cases, e.g. those that are less frequent or not normal
- **postconditions**: the conditions that must hold after the use case finishes

SysML – use cases – descriptions

the use case diagram by itself gives very little information; it must be accompanied by a use-case description that includes:

- **name:** Enter vehicle
- **primary actor:** vehicle occupant (vo)
- **supporting / secondary actors:** vo already in the vehicle
- **preconditions:** there is a free seat in the car
- **main scenario:** Vo opens a door behind which there is an empty seat. Vo then sits down in the empty seat.
- **alternative scenarios:**
  - There is no empty seat directly behind a door. One of the other vo's who is already inside the vehicle needs to shift to the center position in the back seat to make space.
- **postconditions:** the vo is seated in the car

REQUIREMENTS & PACKAGES

SysML – packages – what & when

a model can contain millions of model elements

- each model element has a container/parent
- packages are named containers and are used to organize model elements in a hierarchy

model organization happens through the whole model lifecycle

- having a well-organized model is very important

possible organization principles

- system hierarchy (system, component, sub-component, …)
- process lifecycle (requirements, architecture, design, …, operation)
- teams/disciplines working on the system (software, mechatronics, physics, …)
- model-element kind (behavior, structure, requirements)
- …

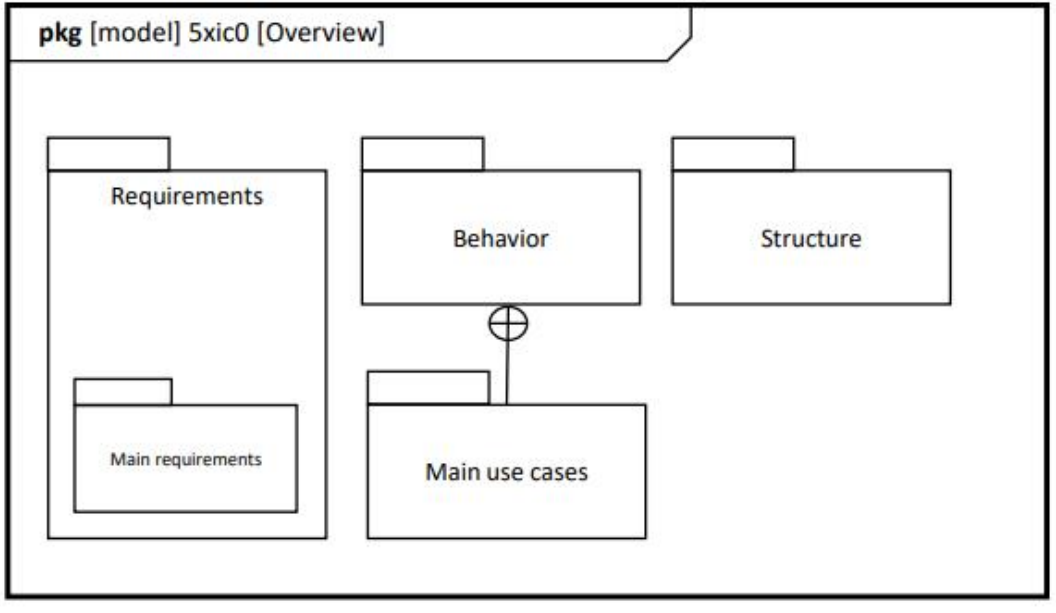
SysML – packages – model elements & diagram (pkg)

model elements

- package: a named container for model elements

relations

- containment
  - by nesting
  - by containment link



benefits of well-written requirements

good systems requirements benefits all subsequent phases of the life cycle:

1. assurance to the customer
2. early bidirectional feedback between customer and engineers
3. method to identify problems and prevent misunderstandings
4. basis for system validation
5. protection for engineers
6. support for planning design
7. assessing the effects of (inevitable) requirement changes

SysML – requirements – what & when

requirement: a condition or capability needed by a user to solve a problem or achieve an objective

- come from many sources, e.g., in the car example
  - customers (top speed at least 130 km/h)
  - developing organization (re-use an earlier developed engine type)
  - government (emission control and safety)
  - …

- systems-engineering challenge to make requirements
  - consistent (not contradictory)
  - feasible (can be realized)
  - sufficiently complete and validated (they reflect all stakeholder needs)
  - verified (system design and realization satisfy them)

SysML – requirements – model elements

model elements

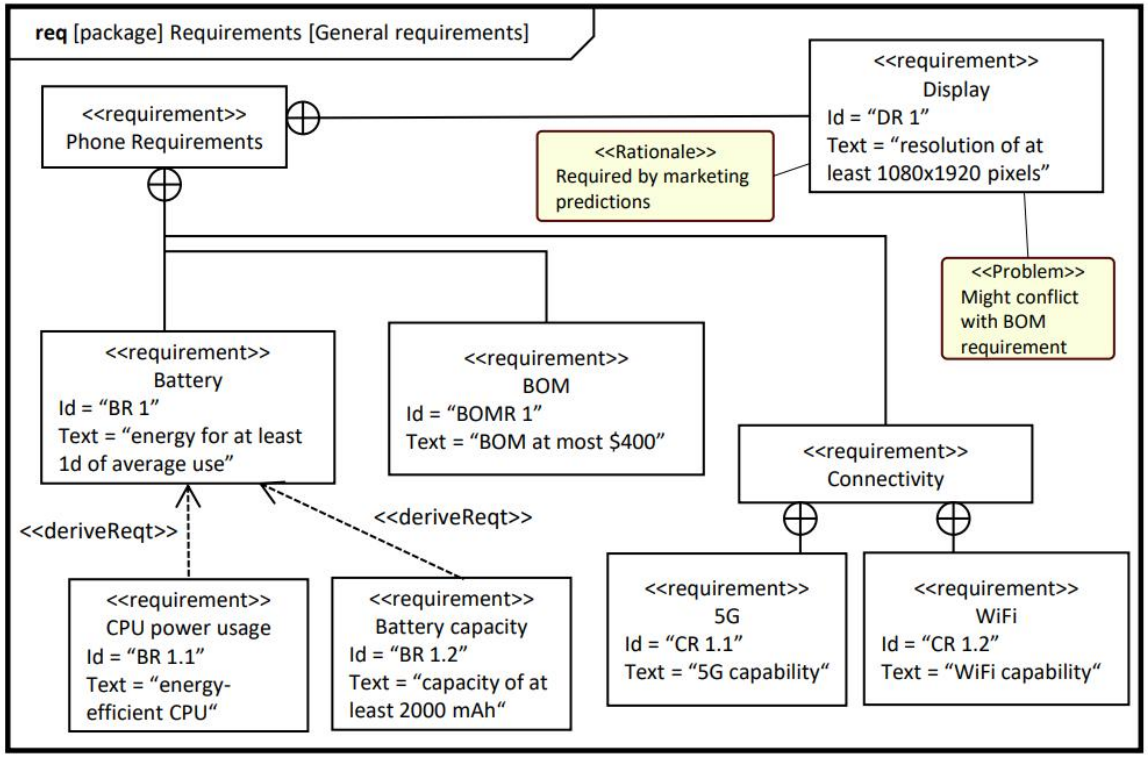
- requirement: name, identifier and textual description of the requirement
- rationale: reason for a particular decision
- problem: flag design issues

requirement relationships

- containment for decomposition of requirements
- deriveReqt for refinement of requirements, based on analysis

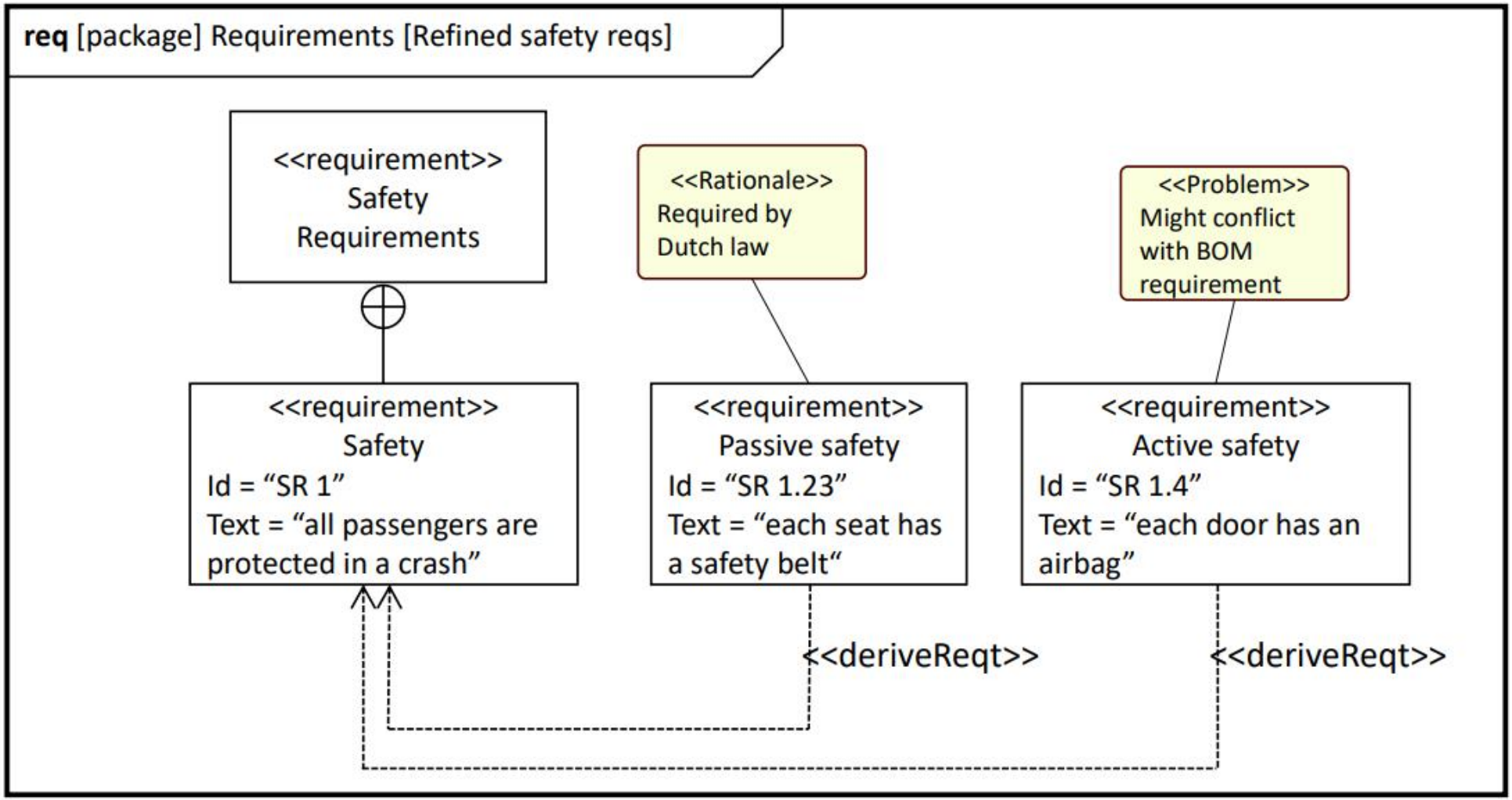
relations to other model elements

- satisfy, verify, refine, trace are cross-cutting and explained in module on allocations



suggested reading:
sections 13.1 – 13.3, 13.6 – 13.10

SysML – requirement diagram (req)



BLOCK DEFINITION DIAGRAM, INTERNAL BLOCK DIAGRAM & PARAMETRIC DIAGRAM

BLOCK DEFINITION DIAGRAM

SysML – blocks – what is it

a block is the modular unit of structure

- physical entity (a system, hardware, …)
- a person, facility (building, road) or entity in the natural environment (atmosphere, ocean, …)
- type of logical or conceptual entity (software, data, …)
- entity that flows through a system (water, current, data, control commands, …)

a block describes a set of instances that share the block's definition

a block has

- structural features that define its internal structure and properties
- behavioral features that define how it interacts with environment and modifies its own state

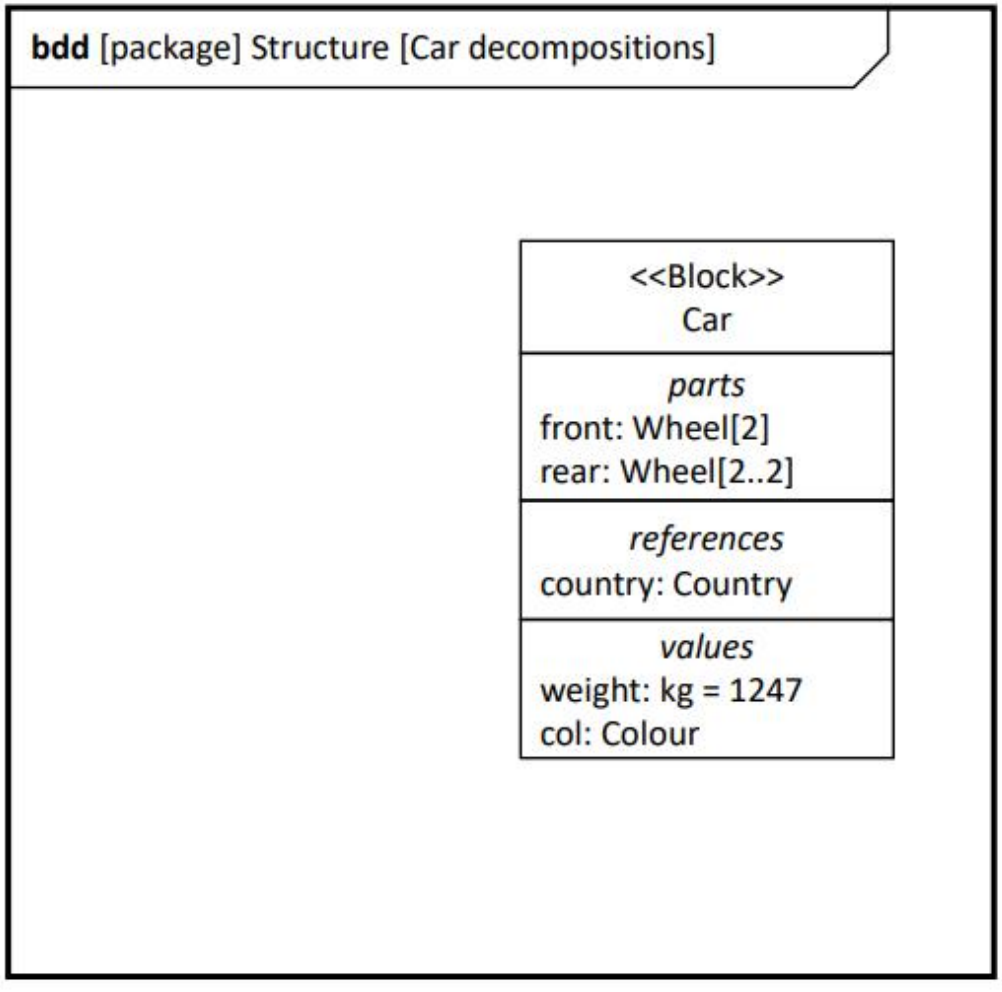
SysML – blocks – model elements

structural features (properties) of a block

- parts: composition relationship between blocks (e.g., a car has two front wheels)
- references: refer to another block instance; a "needs relationship" (e.g., a car is made in some country)
- values: quantitative characteristics of blocks (e.g., weight)
- *ports*: next lecture
- *flow properties:* next lecture

structural features

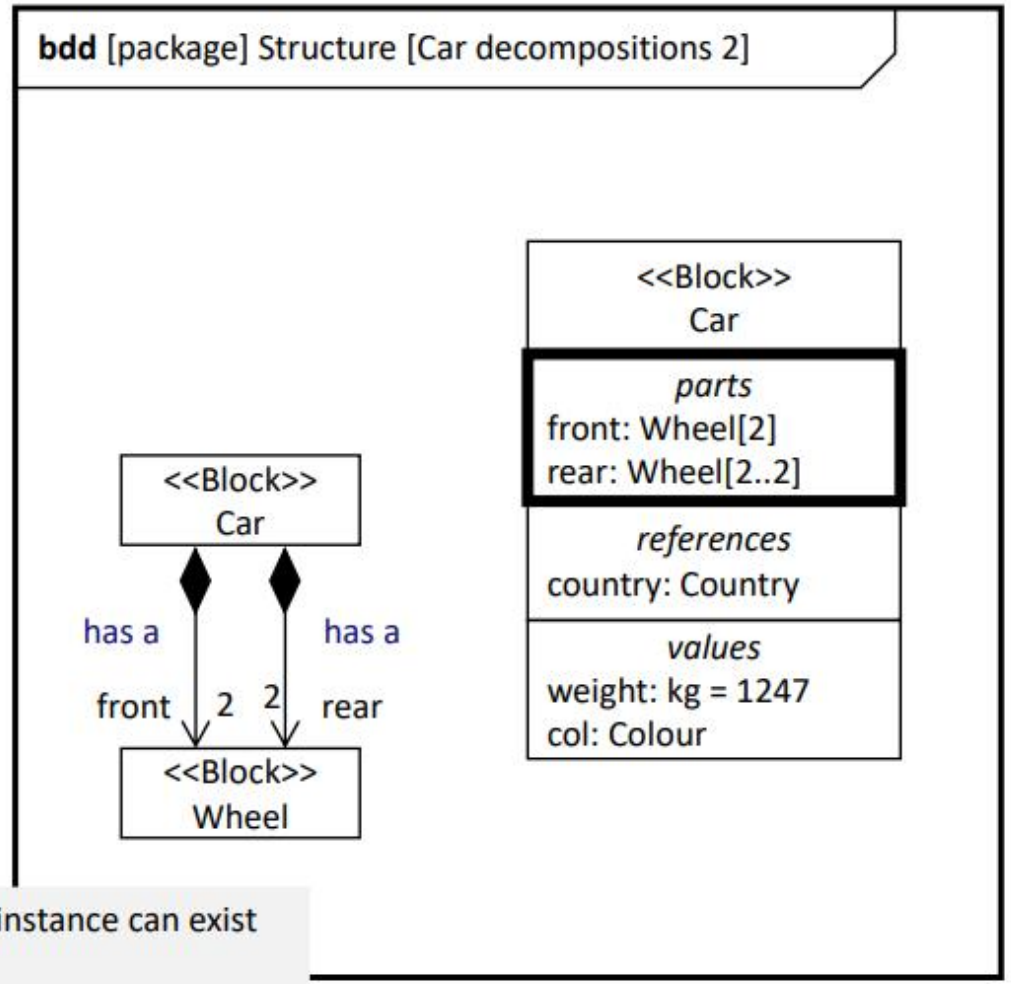
- have multiplicities, i.e. a lower and upper bound, e.g., 1, 0..1, 1..*, …
- can be shown in block compartments



SysML – blocks – model elements – relations

composite association: relates two blocks in whole-part relationship

- line between two blocks with a diamond (whole) and open arrowhead (part); name is optional
- implies a part property in the owning block
- multiplicities:
  - owner: [0..1] (default), or 1
  - part: anything, e.g., 1 (default), 0..1, 1..3, *, …



[0,1] multiplicity on whole end: part instance can exist even if it is not part of a whole

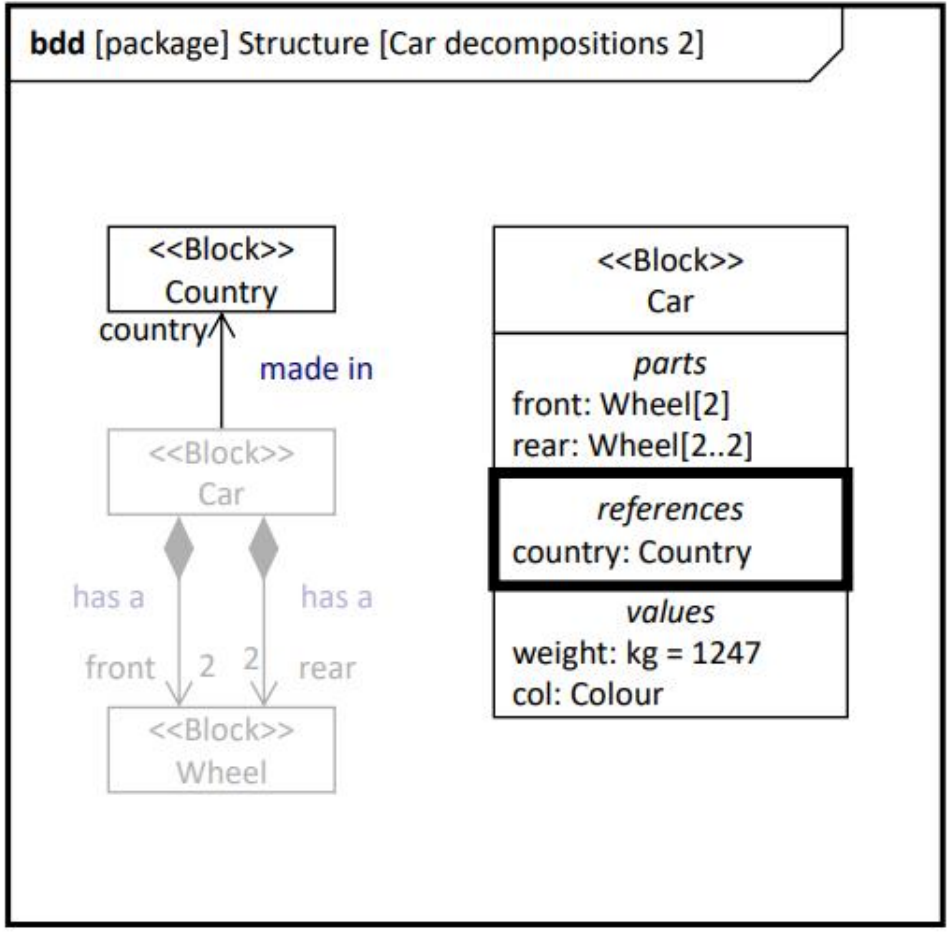
[1] multiplicity on whole end: part instance cannot exist without being part of a whole

ES ELECTRONIC SYSTEMS

SysML – blocks – model elements – relations

reference association: represents a "needs" relationship, not ownership

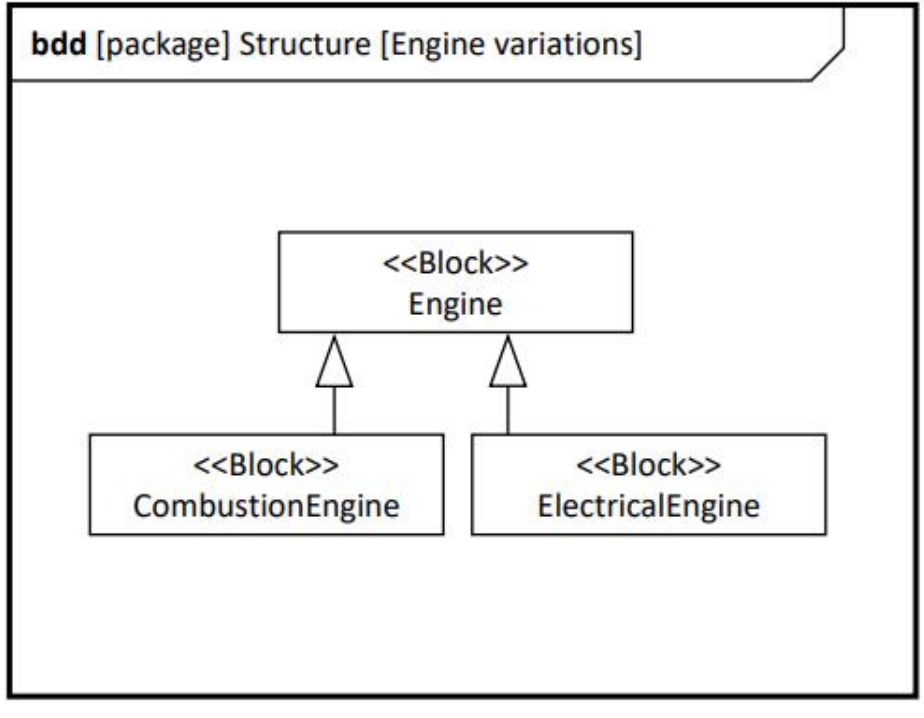
- line between blocks with an open arrowhead (unidirectional), or no arrowhead (bidirectional)
- implies a reference property in the blocks(s)
- multiplicities free (default is 1)



SysML – blocks – model elements – generalization

specialization/generalization: relates blocks in a classification hierarchy

- a classifier is a type (a block) that may be used as the basis for more specific types
- a general classifier contains features that are common to more specialized classifiers
- a more specified classifier (subclass/subtype) inherits the features of the more general classifier (superclass/supertype)



line between two blocks with a hollow arrowhead (more general type): *no multiplicities*

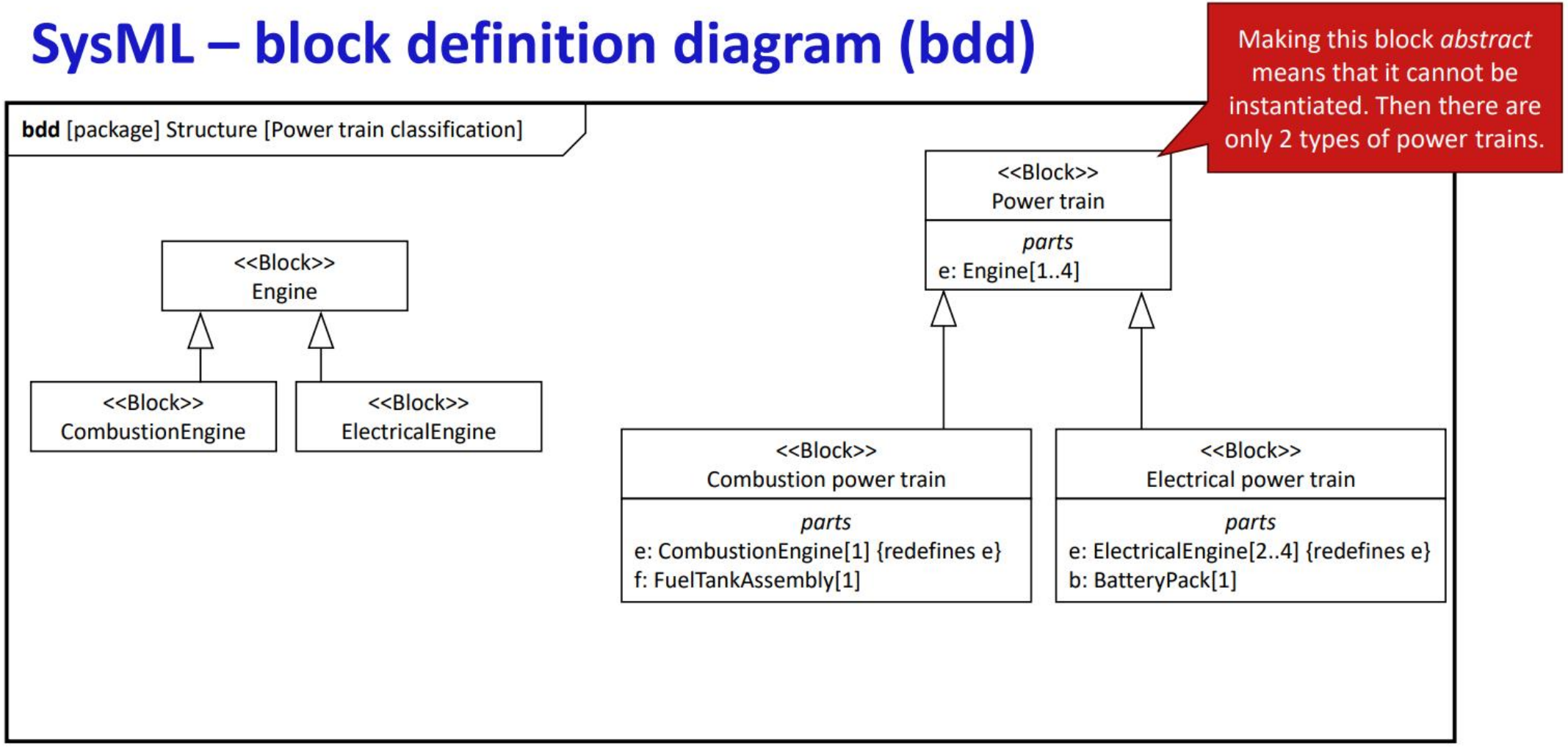
SysML – blocks – model elements – generalization

specialization adds features

specialization can redefine features

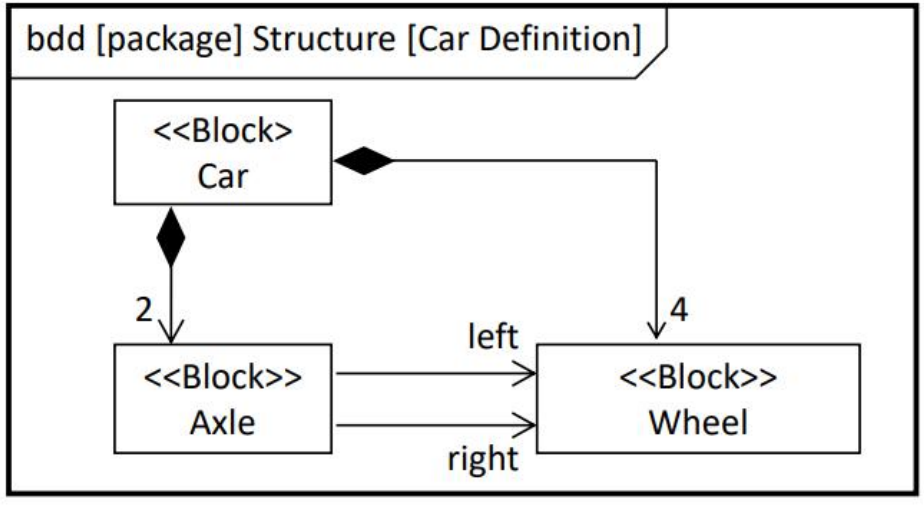
- restrict multiplicity
- restrict the type of the feature
- add or change a default value

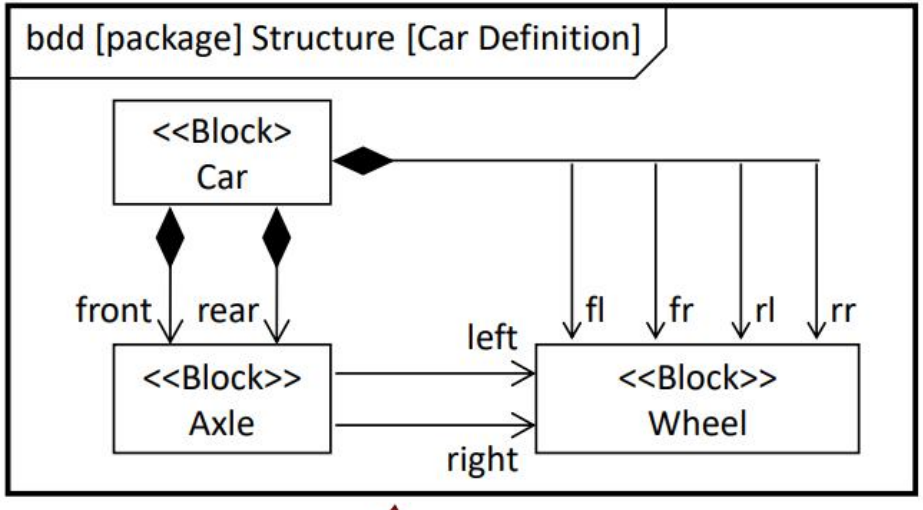
SysML – block definition diagram (bdd)



INTERNAL BLOCK DIAGRAM

SysML – internal block diagrams – modeling internals





Wheels all have a different role

Do this if you want to be able to refer to the wheels individually in your ibd

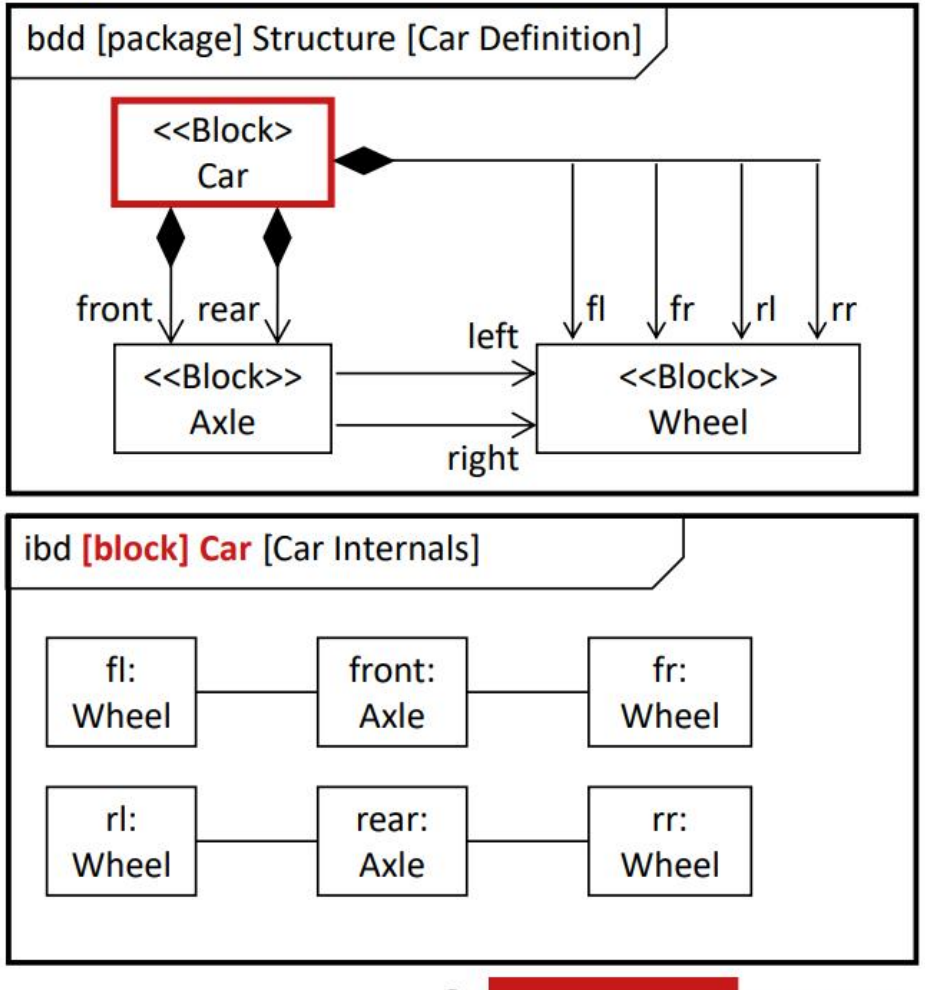
SysML – internal block diagram (ibd)

internal block diagram

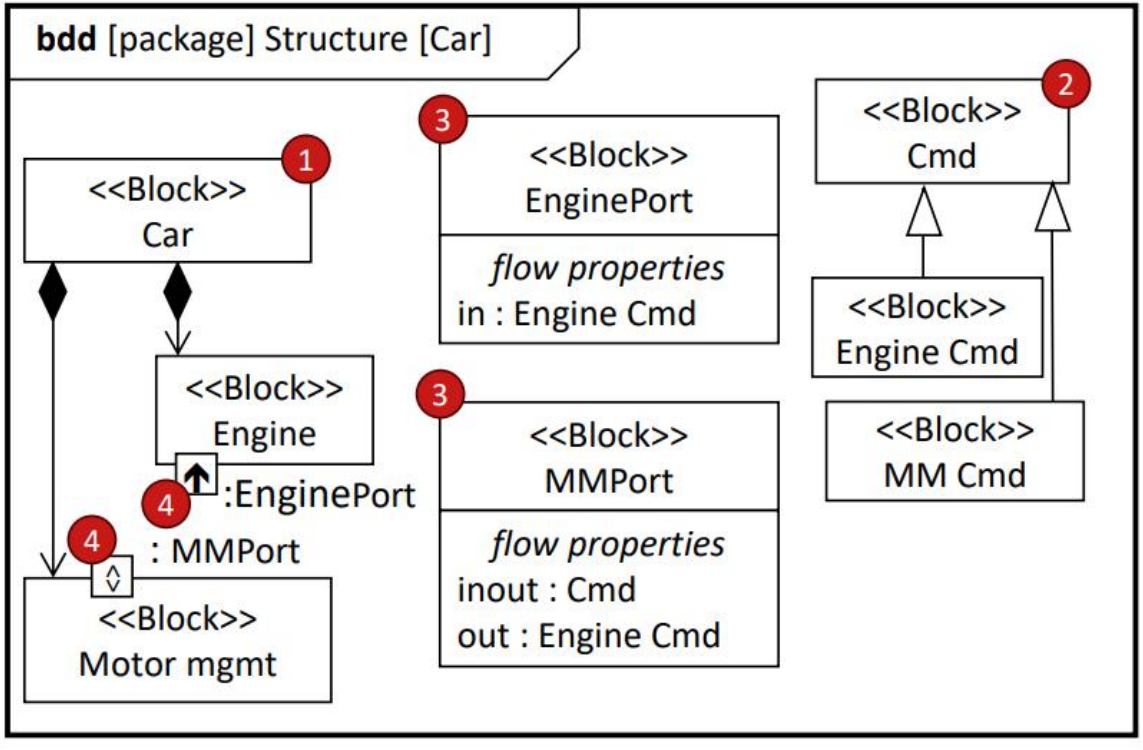
- context is always a single block
- parts or references are shown inside the block and can be connected
  - connector models that there is a connection/communication between two features

used for

- modelling internal block structure
  - parts are shown as solid blocks (can be nested)
  - references are shown by dashed blocks
  - can be connected
- detailed modelling of interfaces and flow (e.g., fuel, electricity, data, pieces, …) with ports

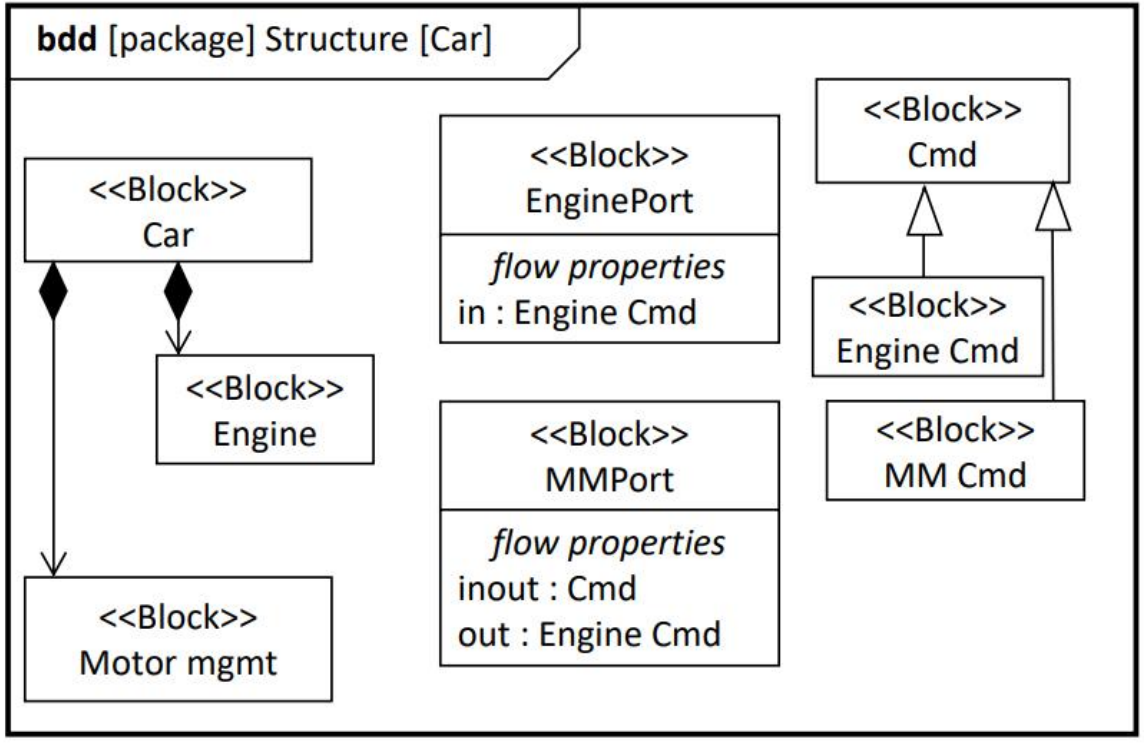


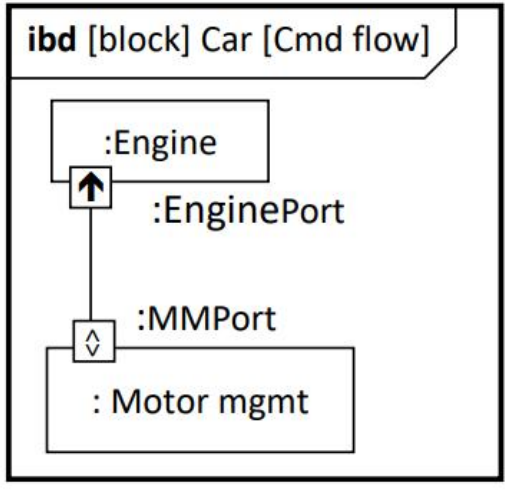
SysML – modeling interfaces and flows – diagrams



1. block and its parts
2. what flows between the parts
3. types of the interfaces (ports)
4. add ports to the blocks
   - arrow indicates the direction of flow properties of the port's type

SysML – modeling interfaces and flows – diagrams





SysML – modeling interfaces and flows – model elements

block features

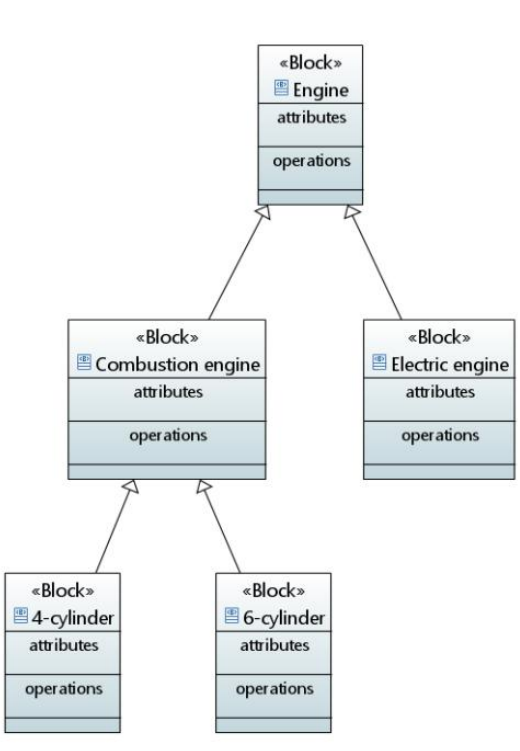
- full ports: access point on a boundary of a block (or on boundary of a part/reference typed by that block)
  - typed by another block
  - which may have flow properties: a special property that specifies *what* flows; has a direction (in, out, inout)

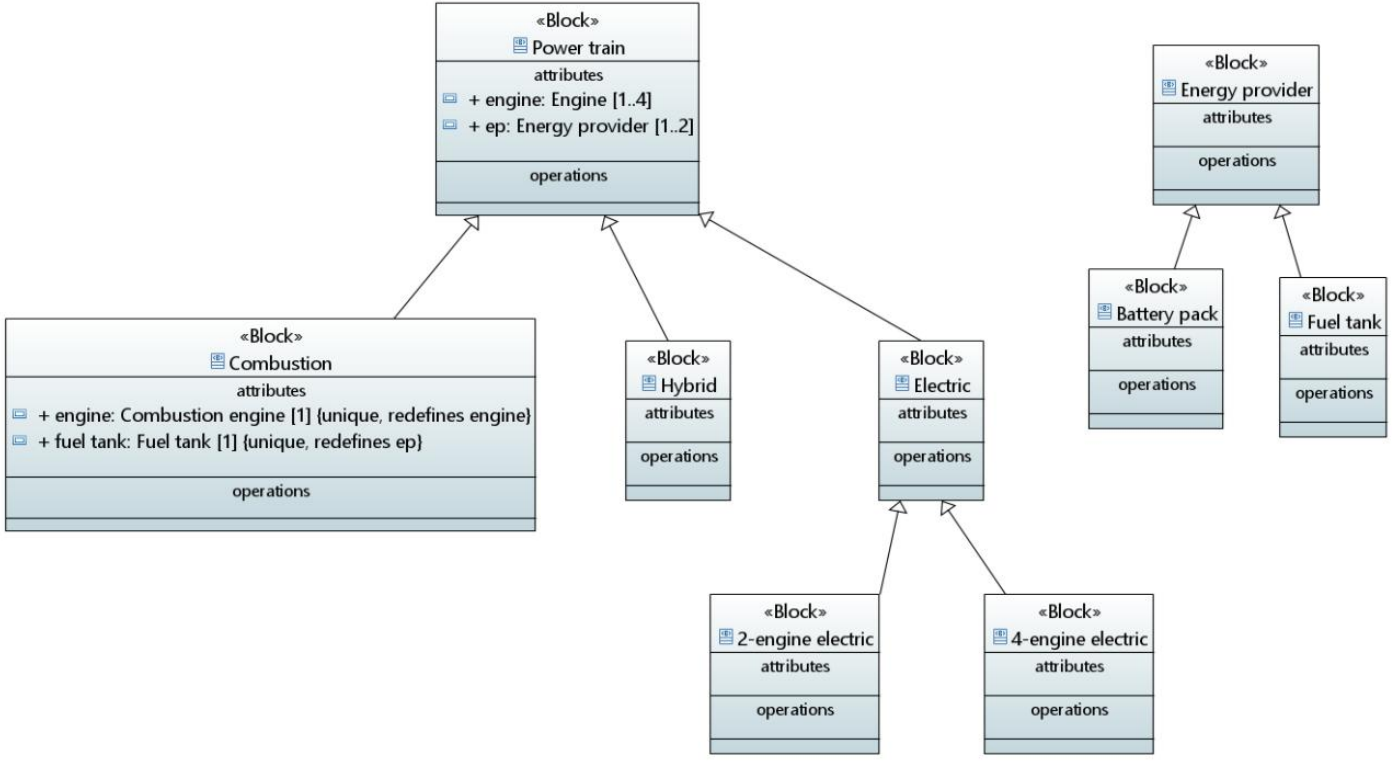
connectors: between full ports and parts on a bdd or ibd

- full ports can be bound to other full ports or parts (constraint on matching flow properties: type & direction)

PARAMETRIC DIAGRAM

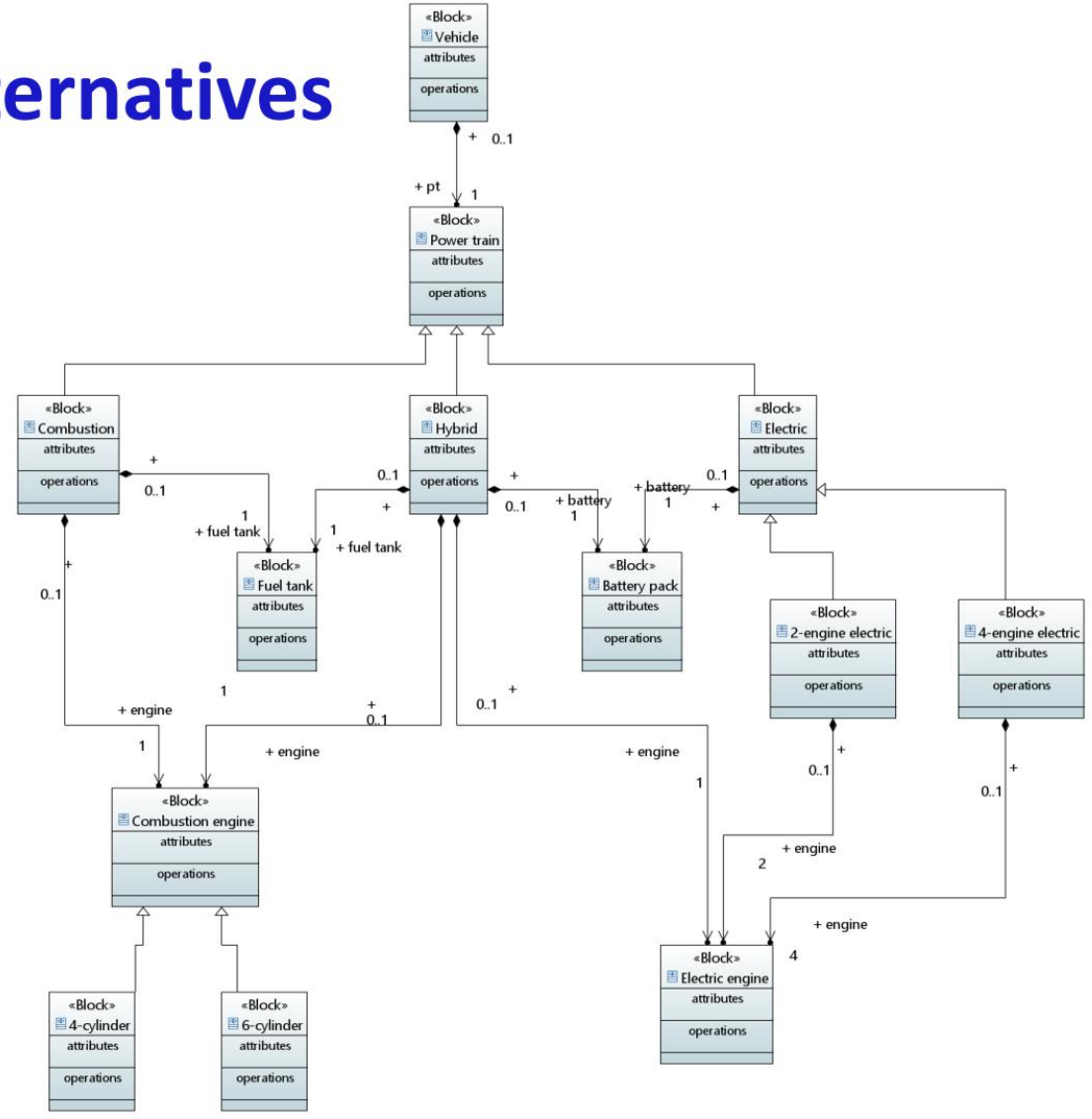
SysML – modeling design alternatives



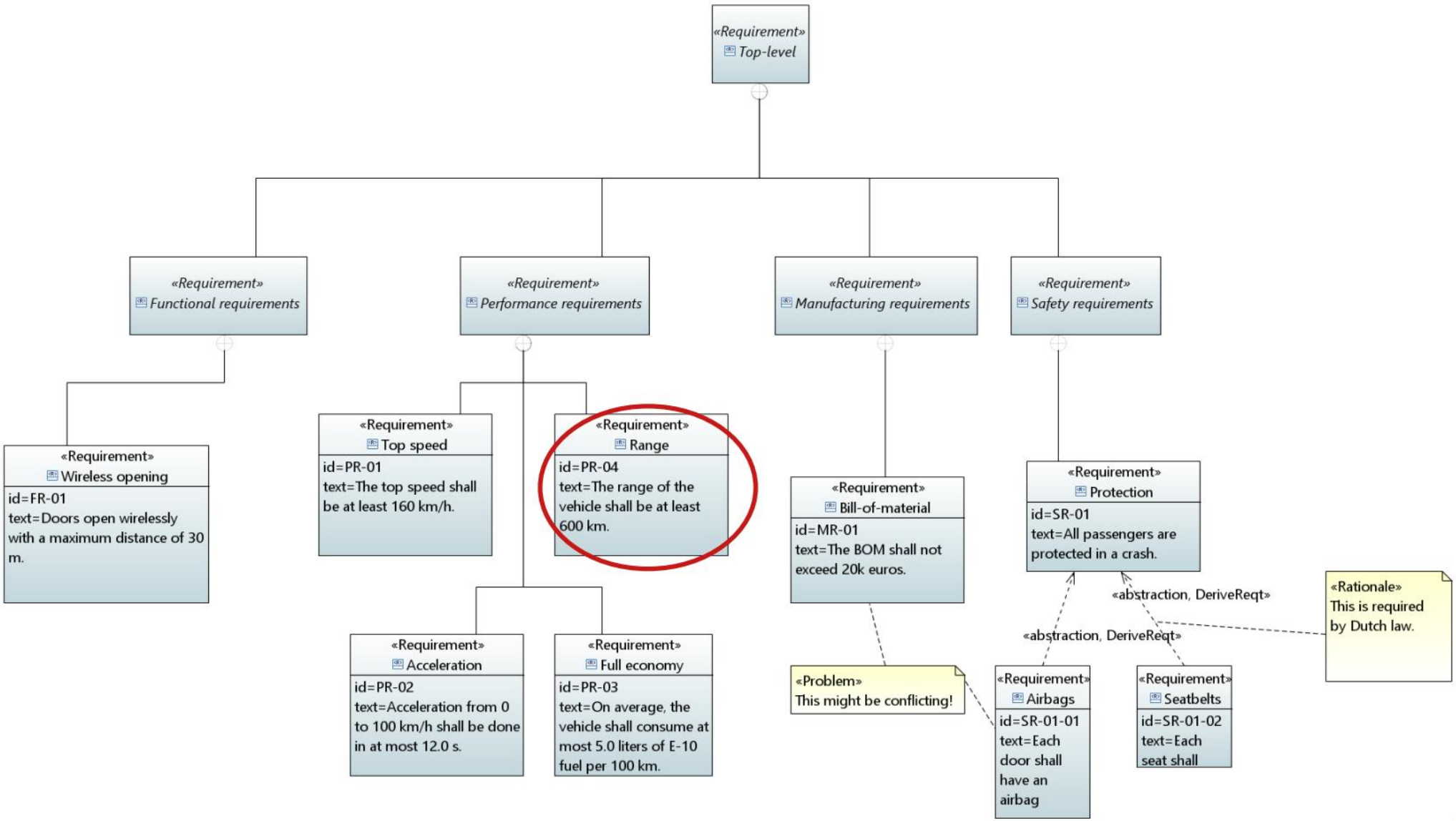


3 classification hierarchies

SysML – modeling design alternatives



SysML – measures of effectiveness (moes)



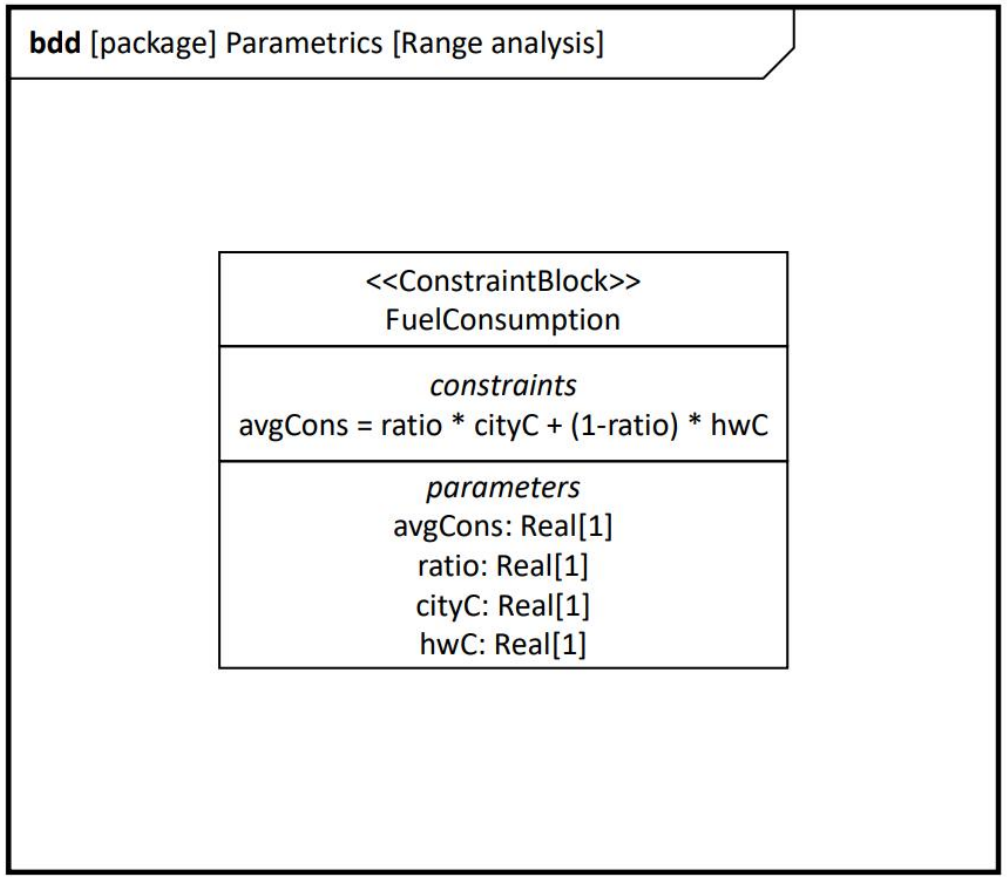
SysML – parametrics – model elements

constraint blocks

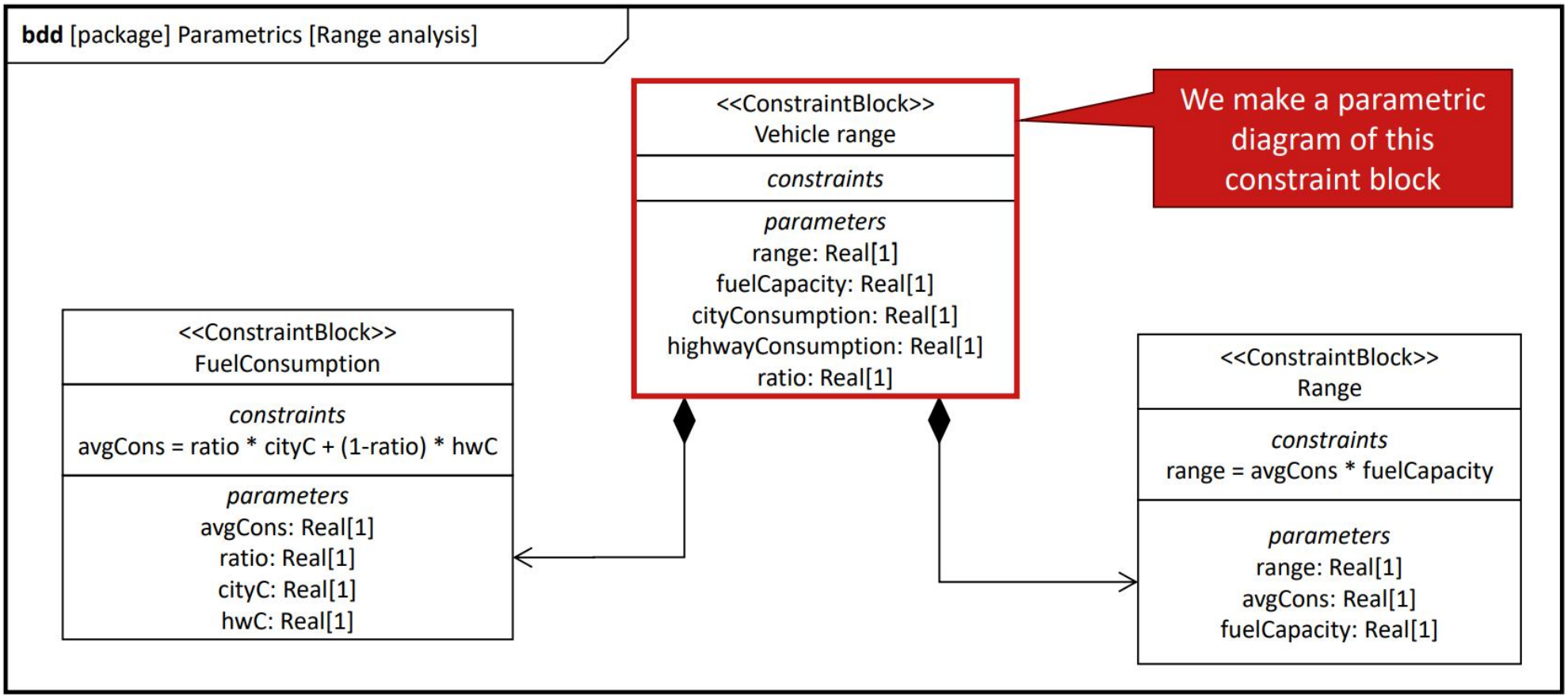
- constraint property (equivalent to a part)
  - expression as a text string using a domain-specific expression language (Java, C, MATLAB, …)
- constraint parameter
  - appears in a constraint property
  - can be bound to other parameters and to (value) properties of the block (binding connector)
  - typically, constraint parameters are value types

constraint blocks can be composed (e.g., on a bdd)

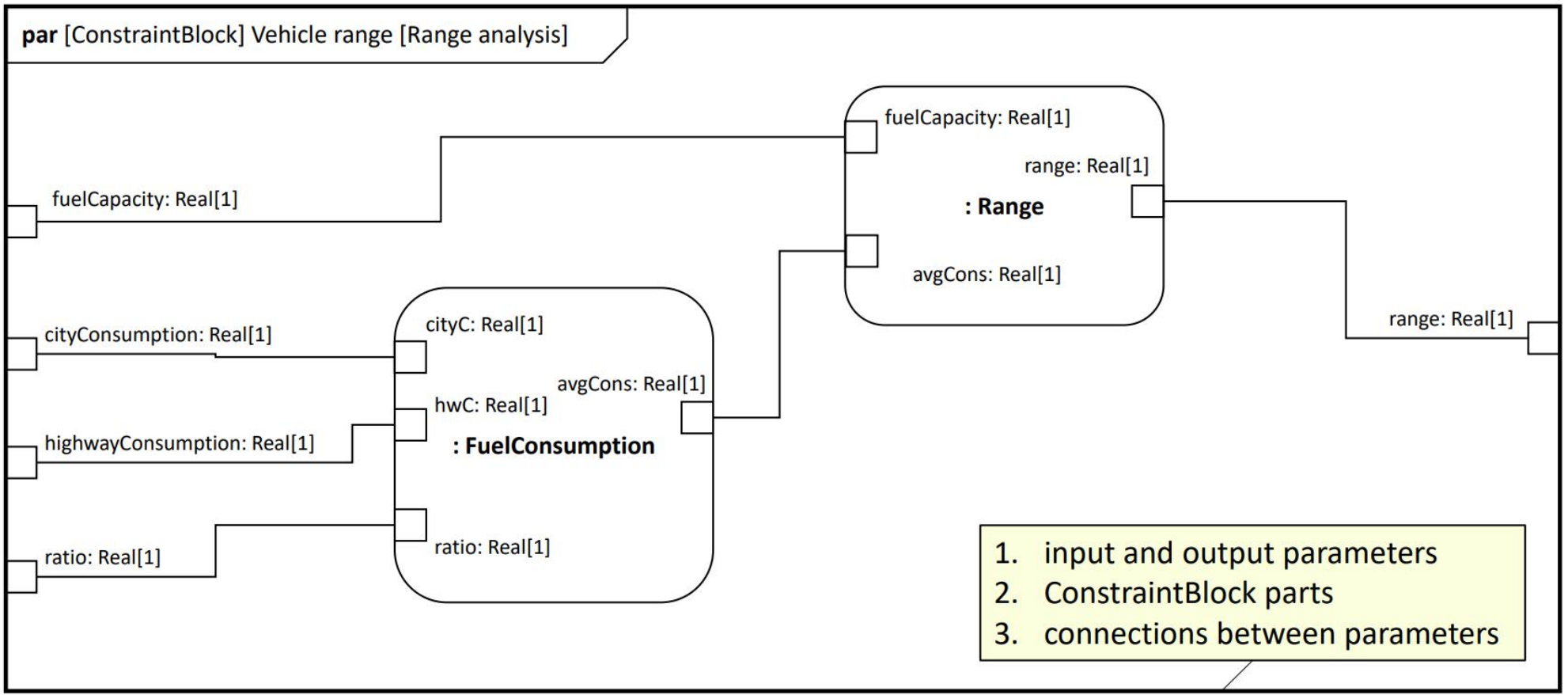
- use composite association to build more complex constraints



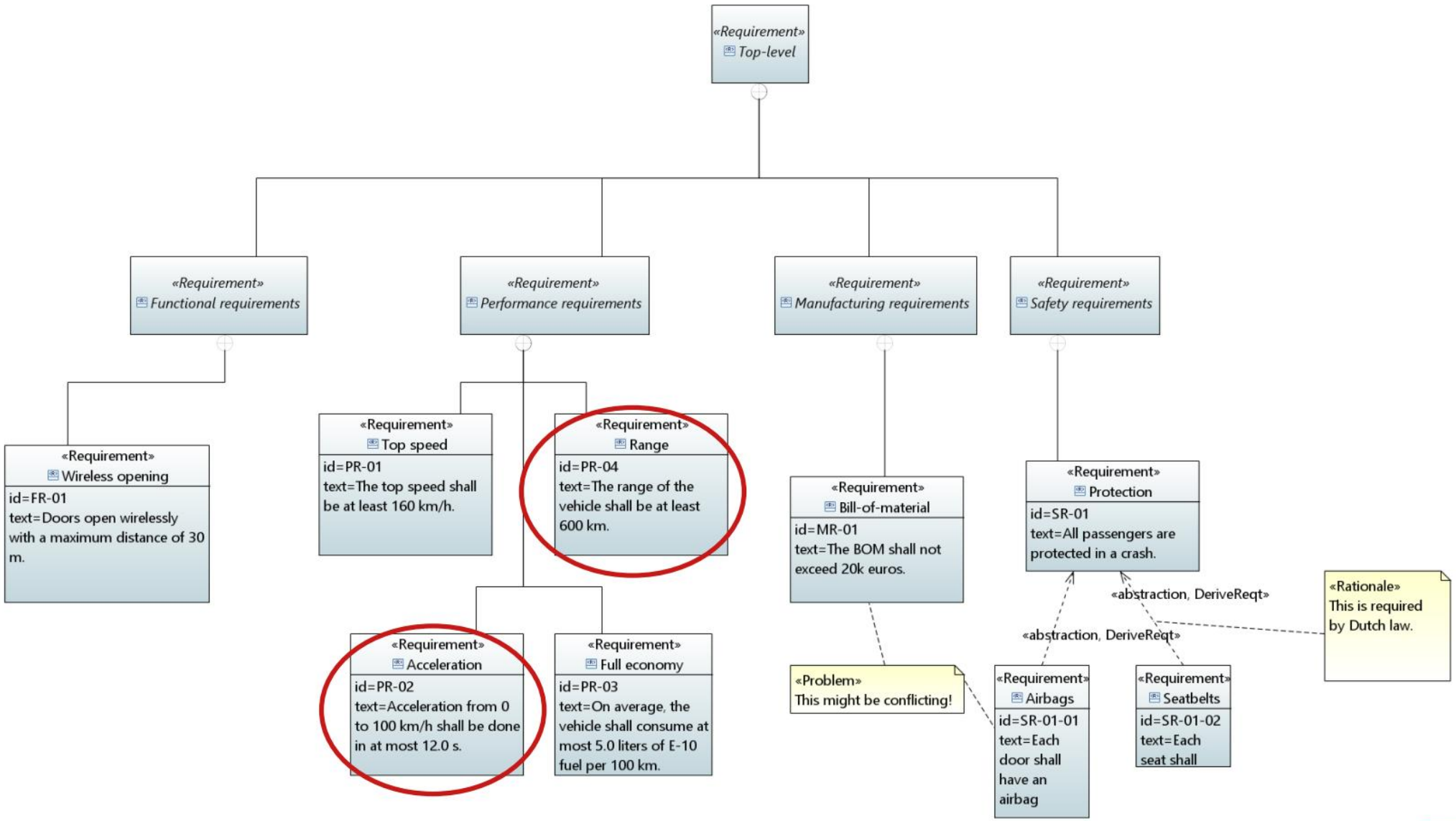
SysML – constraint modeling with a bdd



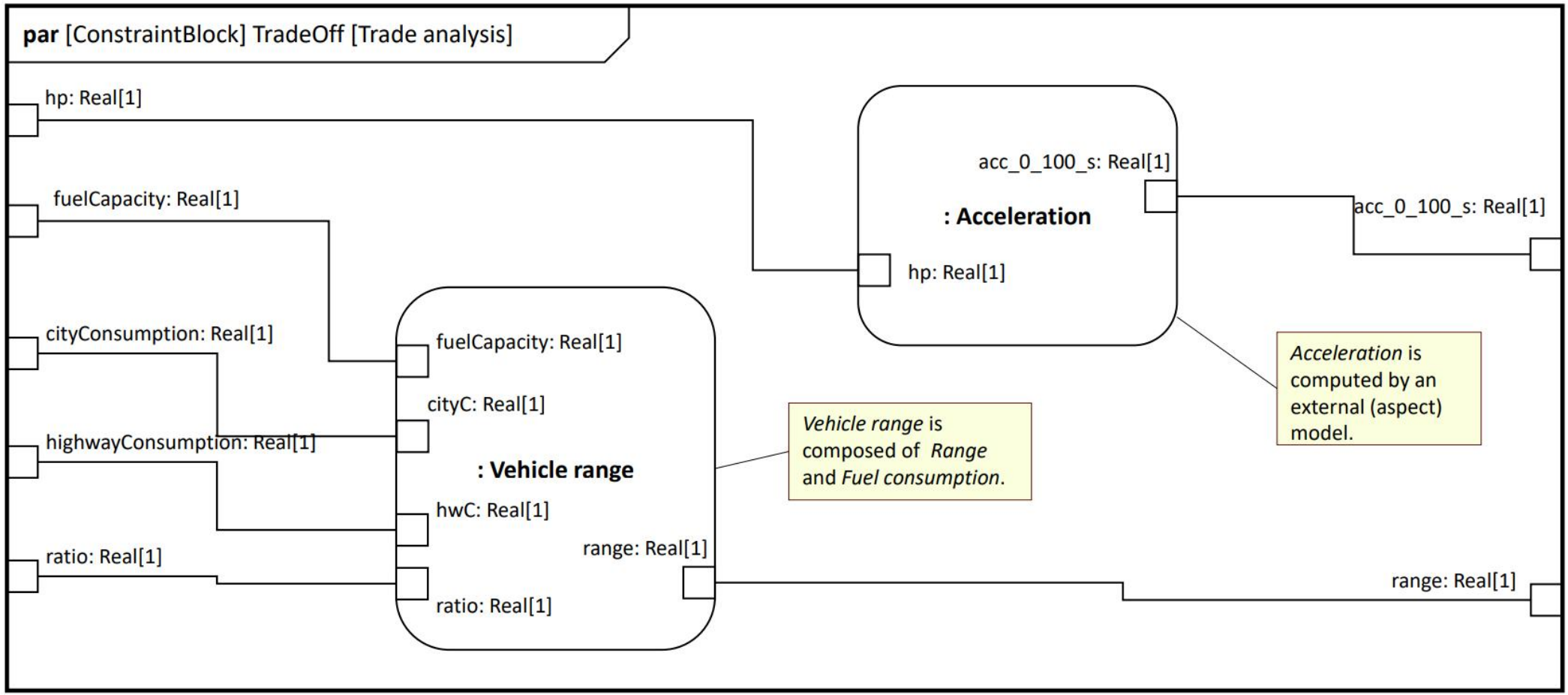
SysML – parametric diagram *(an IBD of a ConstraintBlock)*



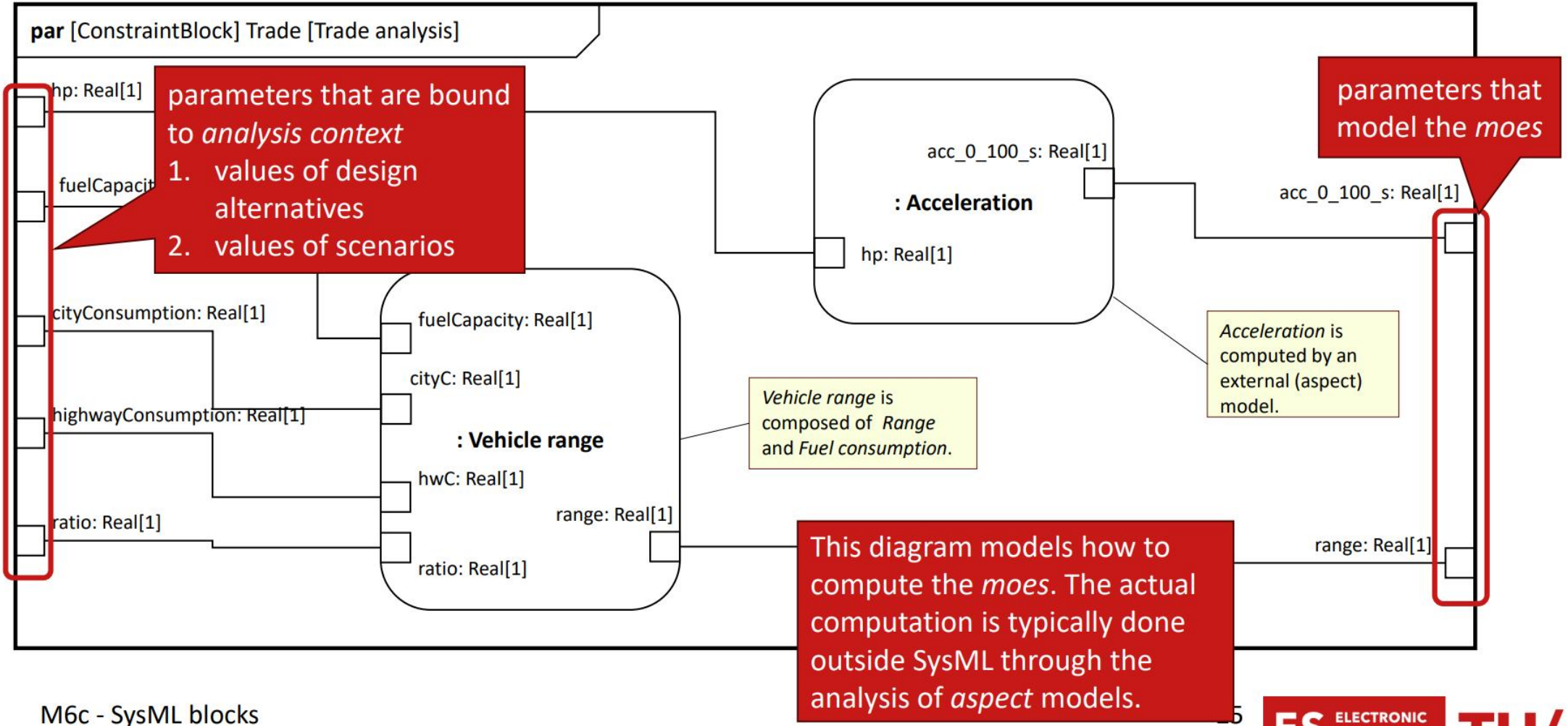
SysML – measures of effectiveness (moes)



SysML – parametric diagram for trade analysis

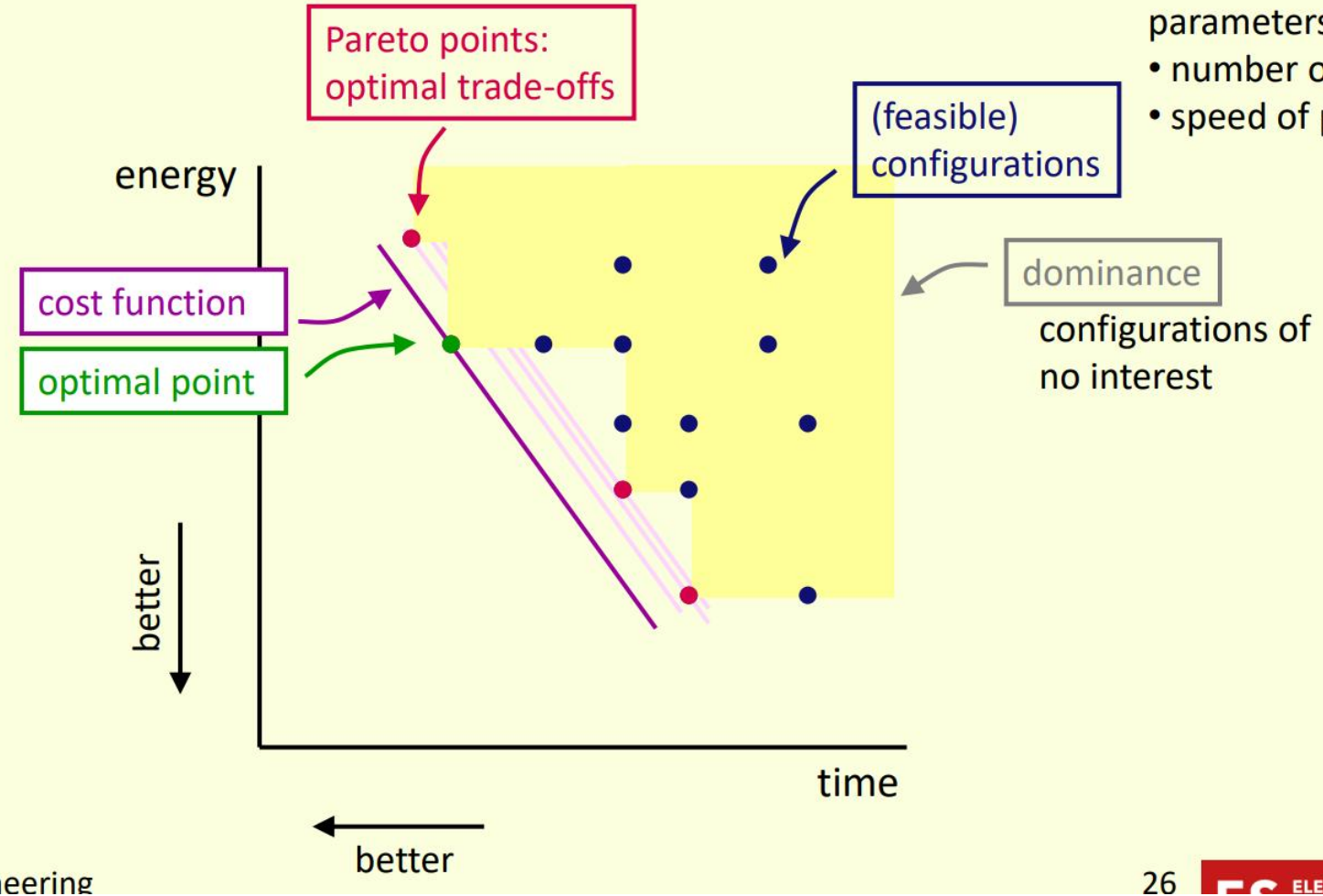


SysML – parametric diagram for trade analysis



optimization - a Pareto space

multiple objectives,
e.g. **time** and **energy**



ACTIVITIES

SysML – activities – purpose

system structure can be modeled with blocks

- logical structure
- physical structure
- decomposition

=> static view

system behavior can be modeled with activities

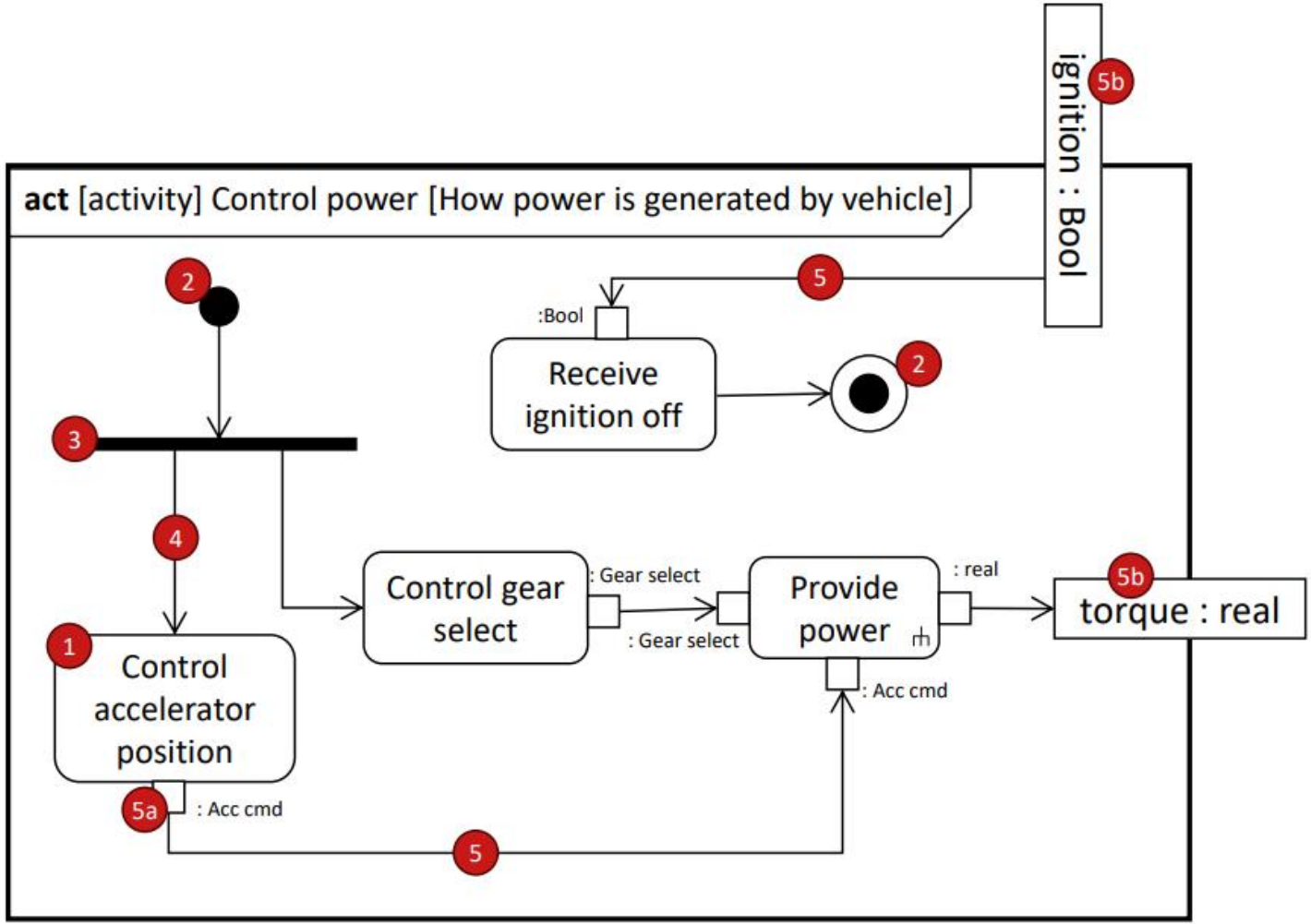
- sequential/parallel and conditional execution of actions
- decomposition

=> dynamic view

SysML – activities

1. actions
2. control nodes
3. routing nodes

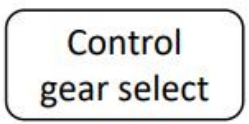
4. control flow
5. object flow
   a. pins
   b. parameters



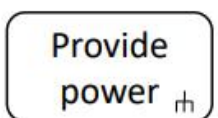
SysML – activities – model elements – actions

types of actions:

- opaque actions (not further decomposed)



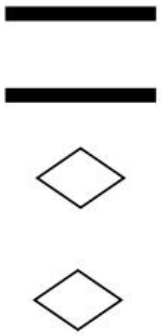
- call-behavior actions (refer to another activity)
  - pin for each parameter of the called behavior (name, type and multiplicity must match)



SysML – activities – model elements – routing/ctrl nodes

routing nodes that modify token streams on both object and control flows

- fork node: one input, at least one output -> replicates input tokens on output
- join node: at least one input, one output -> produces an output when all inputs have a token
- decision node: one input, at least one output -> input token traverses to one output based on condition
- merge node: at least one input, one output -> each input token is immediately routed to the output



control nodes for starting and ending control flows and the owning activity

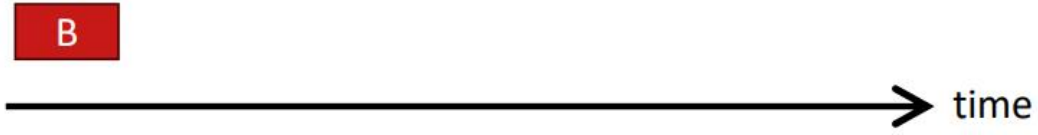
- initial node: when activity starts, a control token is placed in each initial node
- activity-final node: termination of activity
- flow-final node: control-token sink

SysML – activities – semantics

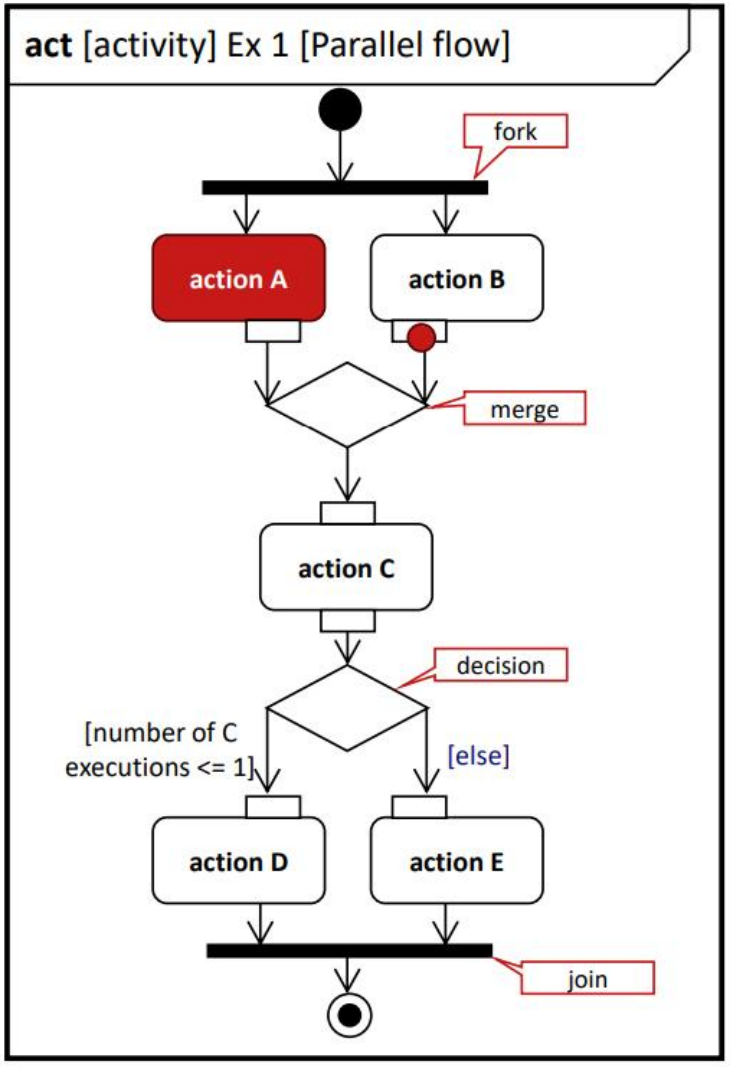
token-flow semantics related to Petri-Nets

- control tokens flow on control-flow edges
- object tokens flow between parameters and/or pins of actions on object-flow edges
  - parameters and pins have a multiplicity and type
  - they can store/buffer object tokens (of the specified type)
- an action can execute if
  - each input pin has at least the minimum required tokens
  - a token is available on each of the incoming control flows
  - these tokens are consumed during execution of the action
- action execution produces tokens
  - one token on each outgoing control flow
  - a number of tokens on each output pin (at least the lower bound of the multiplicity of the pin)

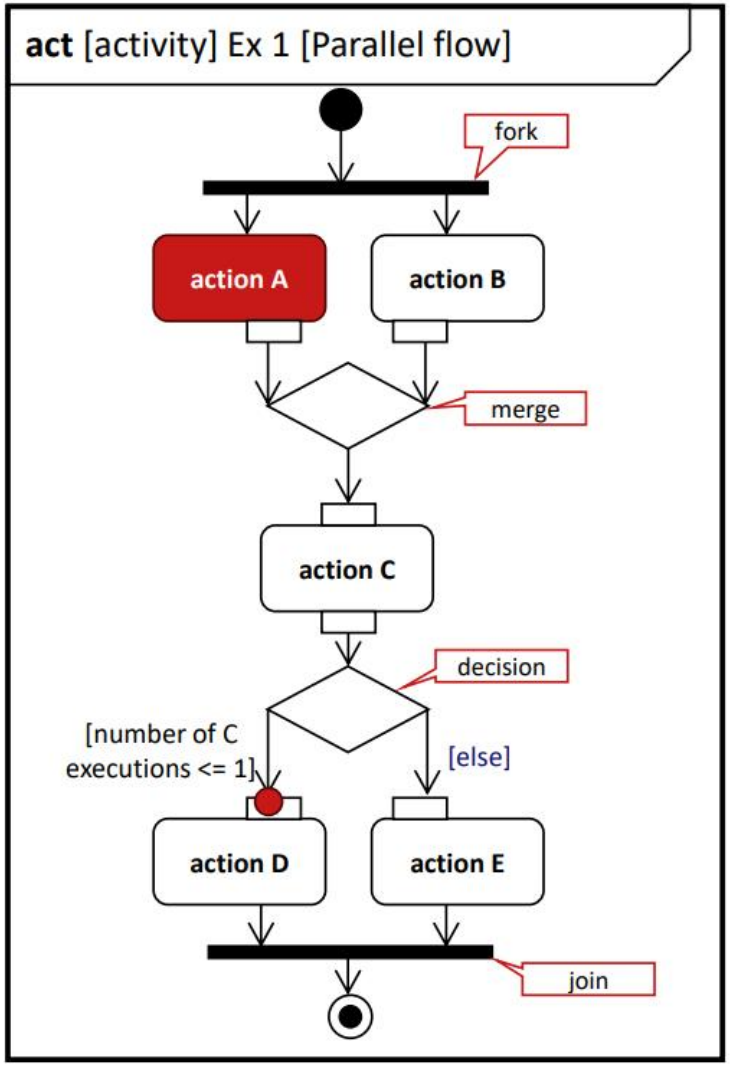
SysML – activities – semantics



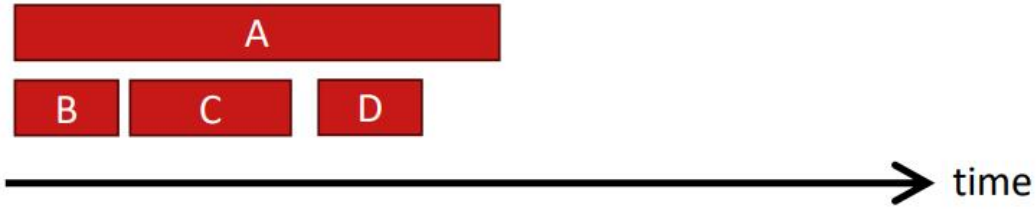
Note that activity diagrams do not have a notion of time. The diagram does, however, specify constraints on the order of events (start and end of actions). A Gantt chart may or may not respect those constraints.

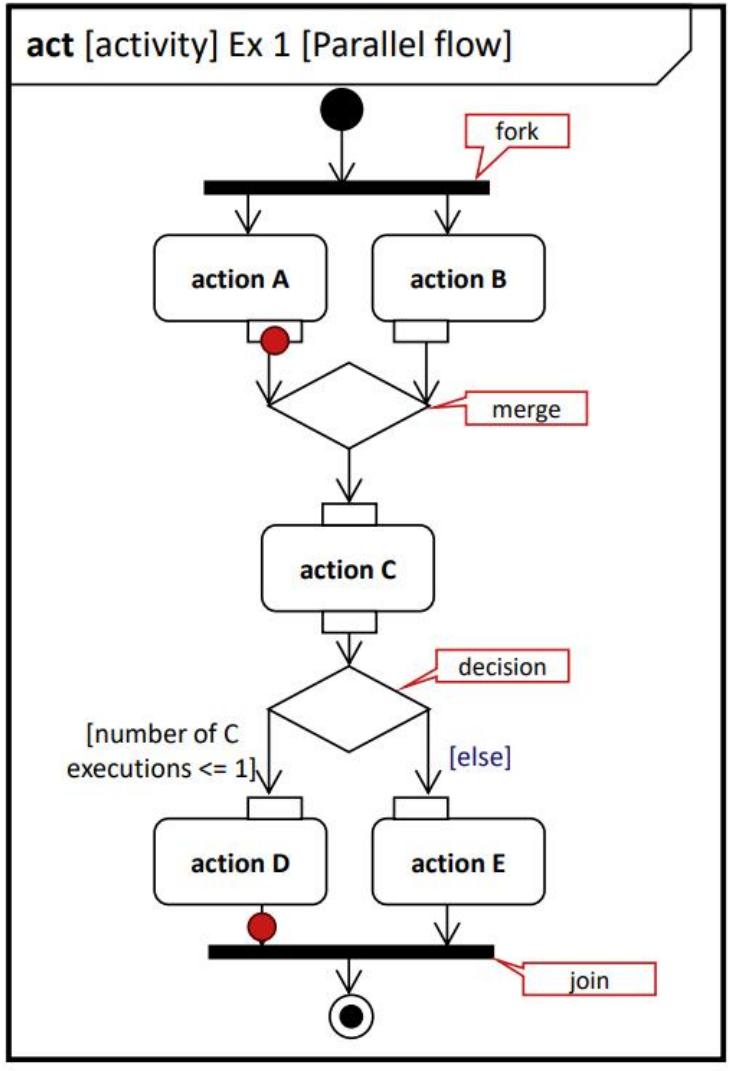


SysML – activities – semantics

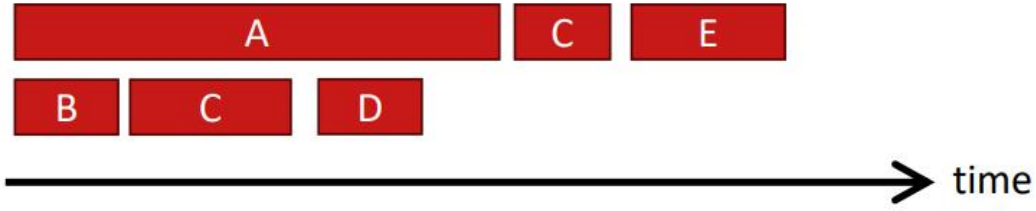


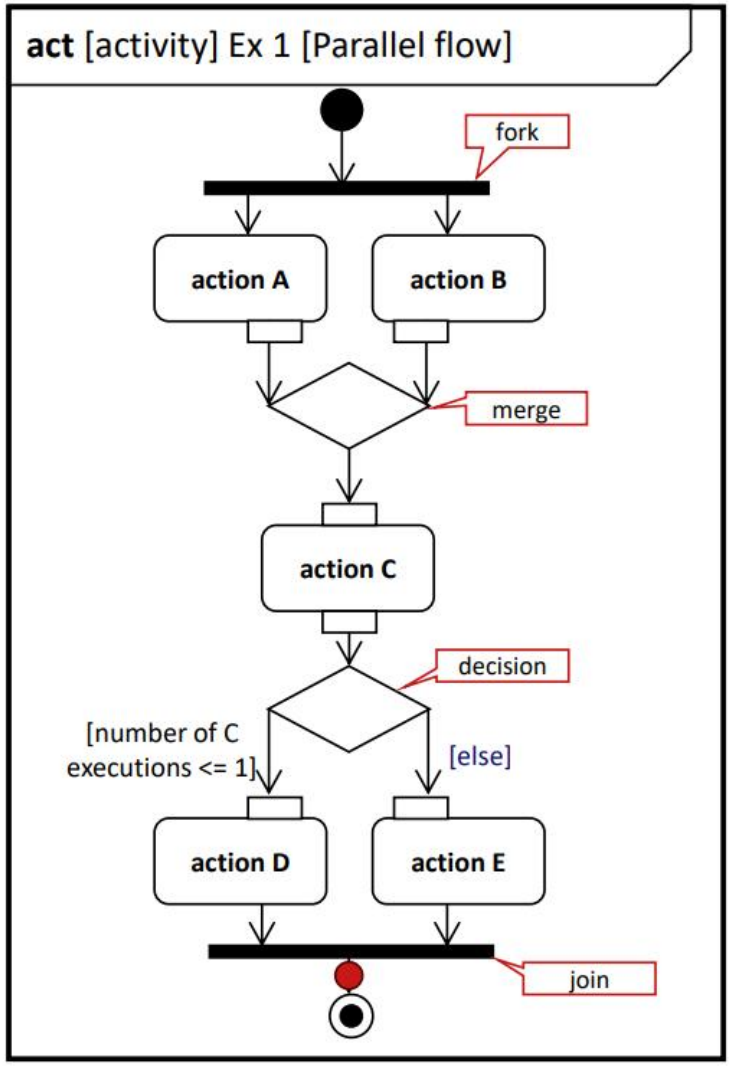
SysML – activities – semantics



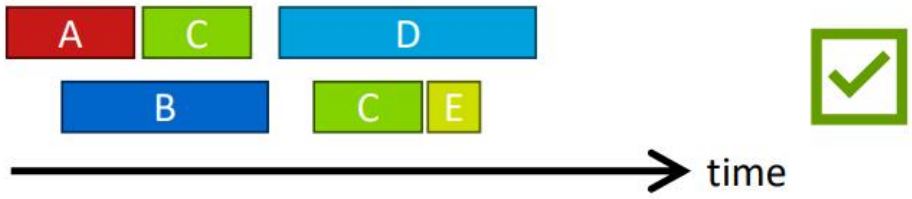


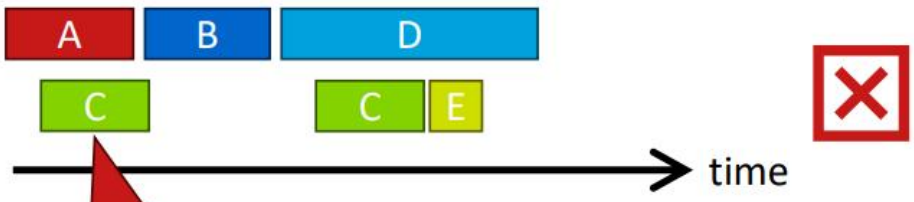
SysML – activities – semantics

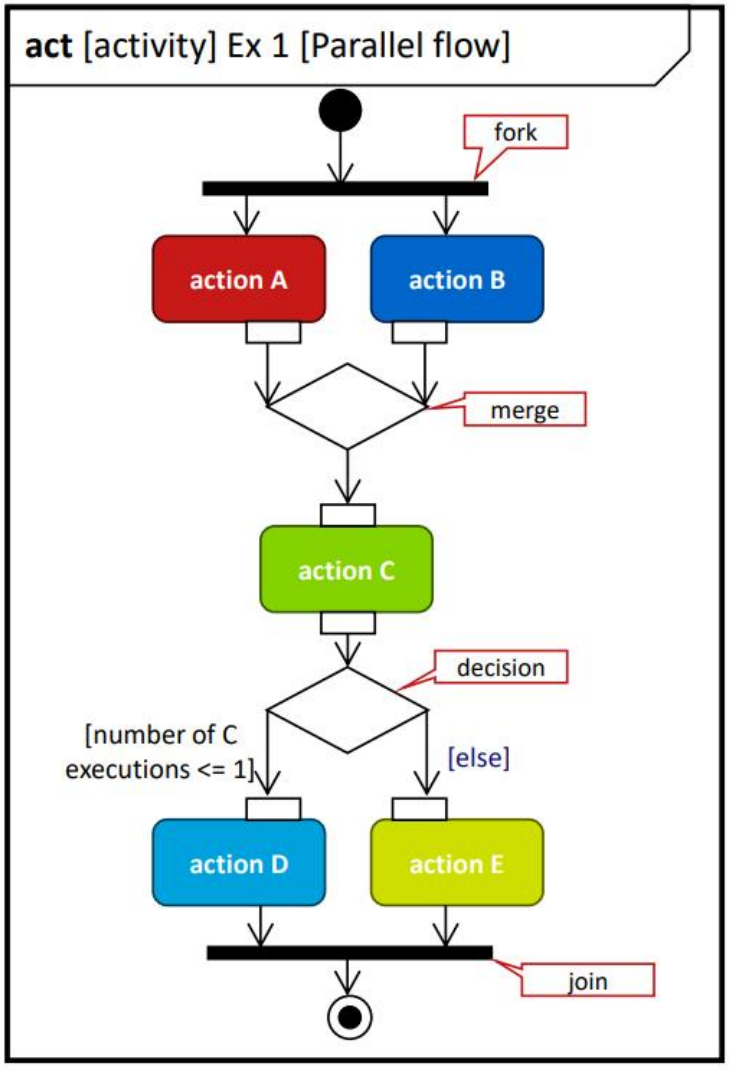




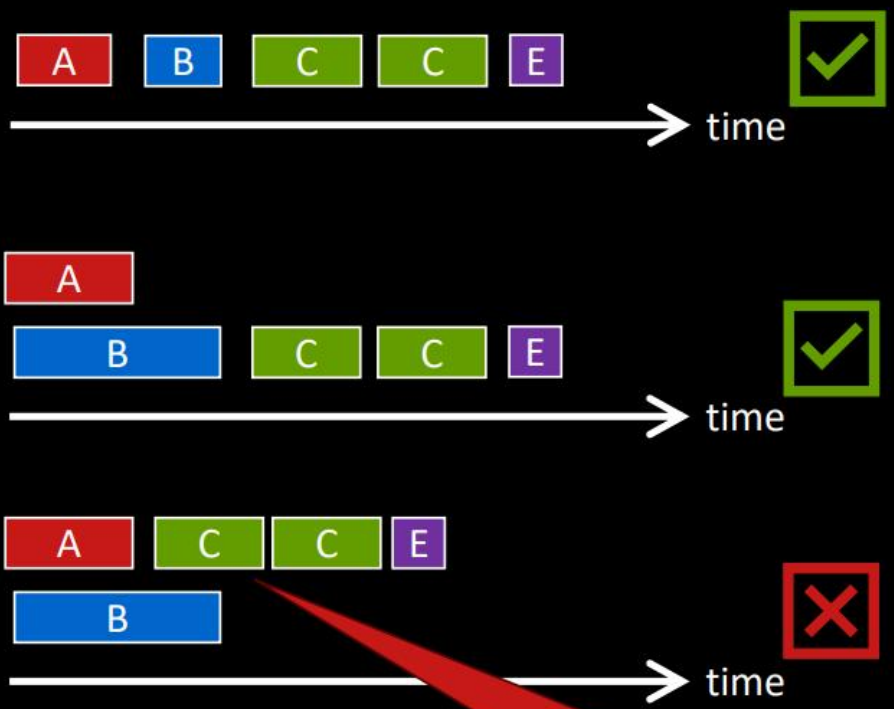
SysML – activities – semantics

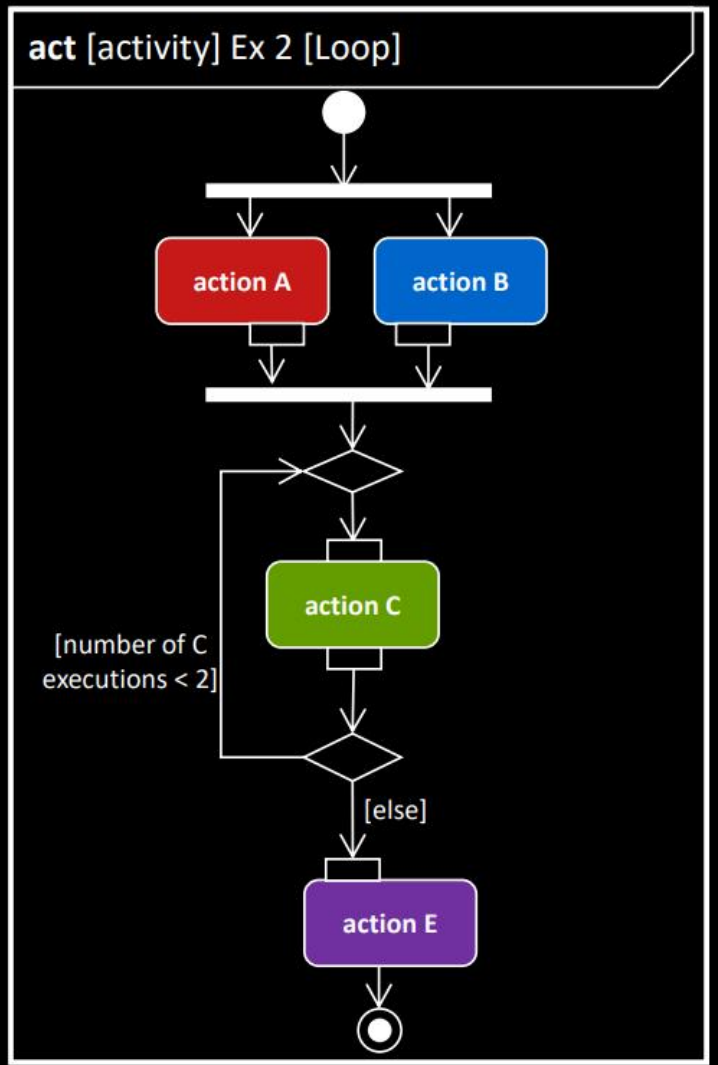






think – pair – share





ALLOCATIONS

SysML – allocations

a system usually has multiple descriptions

- structural / logical structural
  - defining the decomposition in its parts / usages / components
  - bdd, ibd, par
- functional / behavioural
  - defining the decomposition in its sub-functions / sub-behaviours
  - uc, act

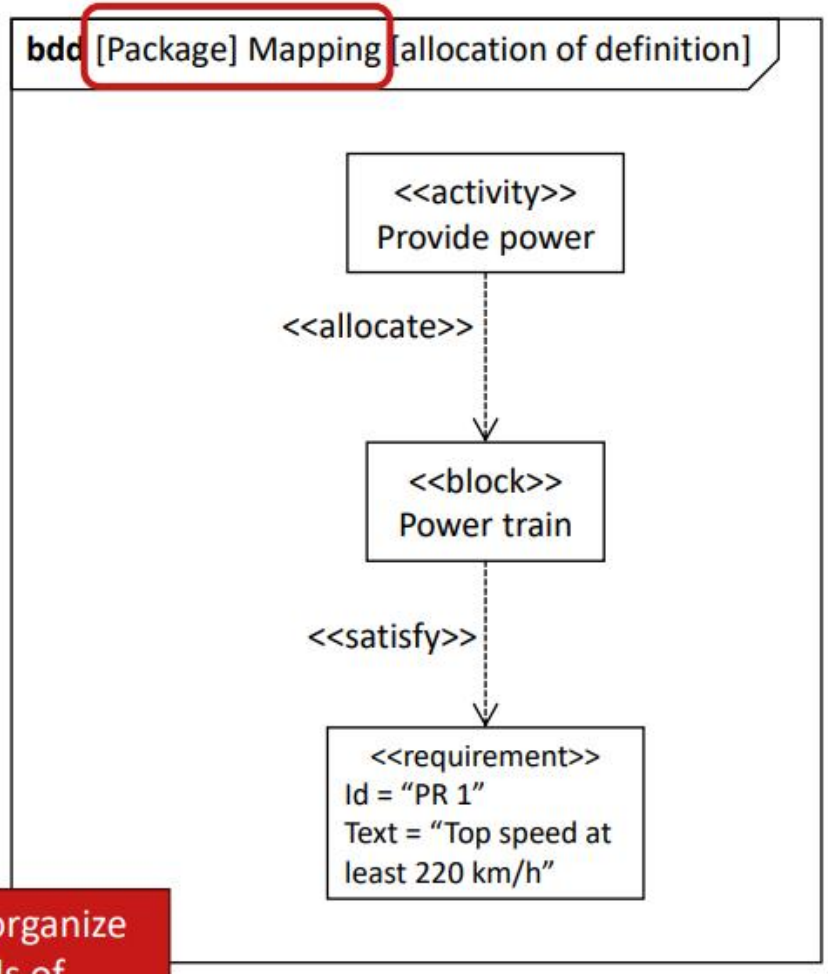
SysML – allocations

allocations define how different descriptions are related

- how are requirements realized by the design
- how is behavior (i.e., use cases and refinements with activities) implemented
- how is a logical architecture implemented by a physical architecture
- …

there is no special diagram (because of cross-cutting nature)

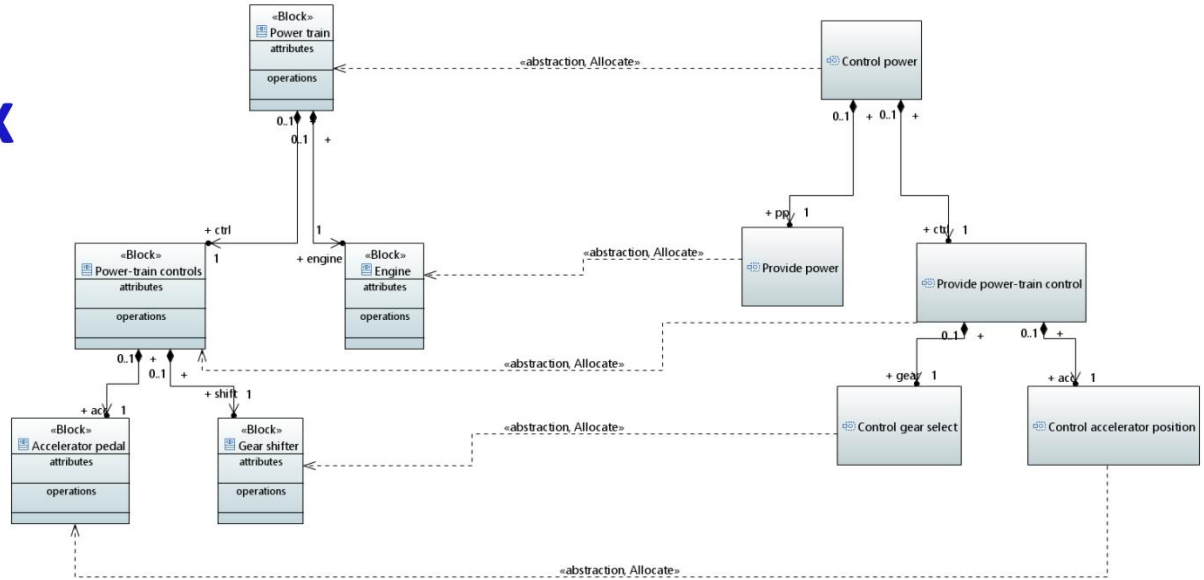
- visualize them in (separate) diagrams; bdd / req
- record them in a table/matrix



In Papyrus, you can organize the different kinds of allocations also in packages

ES ELECTRONIC SYSTEMS TU/e

SysML – allocation matrix

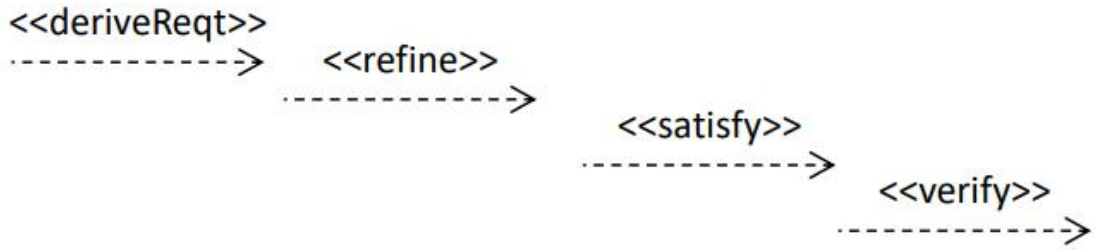


Activity / Block	Power train	Engine	Power-train controls	Accelerator pedel	Gear shifter
Control power	X				
Provide power		X			
Provide power-train ctrl			X		
Control gear select					X
Control accelerator pos				X	

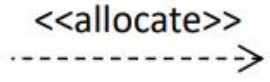
SysML – allocations

we look at three kinds of allocations (all relationships between model elements)

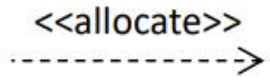
- requirement allocation
  - deriveReqt relationship
  - refine relationship
  - satisfy relationship
  - verify relationship



- functional allocation
  - allocation relationship



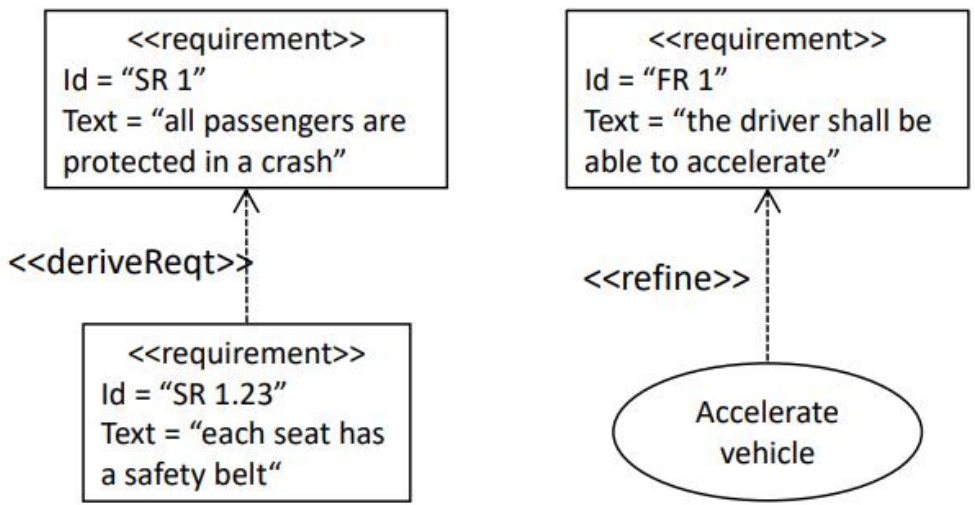
- structural allocation
  - allocation relationship



SysML – requirement allocation

- deriveReqt : requirement to requirement; states that the src requirement is derived from the dst requirement

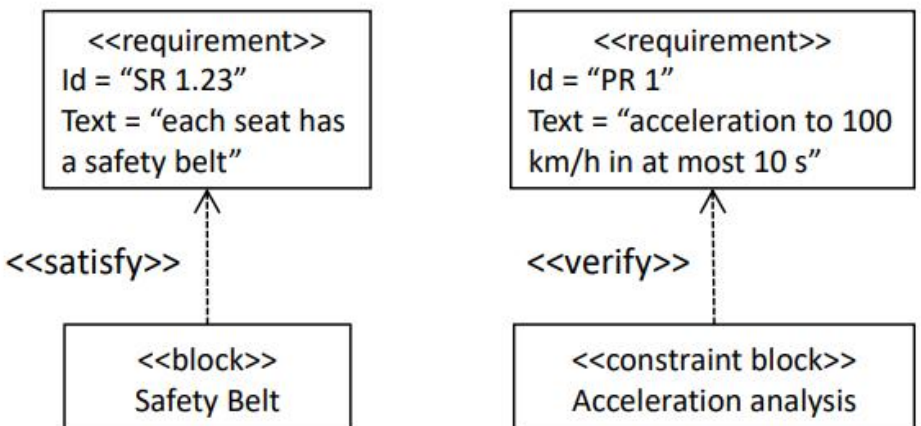
- refine : between a model element and a requirement (can be both ways); reduces ambiguity, clarifies



SysML – requirement allocation

- satisfy : model element (block, activity) to requirement; asserts that the requirement is satisfied by the model element

- verify : constraint block to requirement; proves that the requirement is satisfied by the analysis specified in the constraint block



SysML – functional allocation

decouples form from function: Y-chart pattern

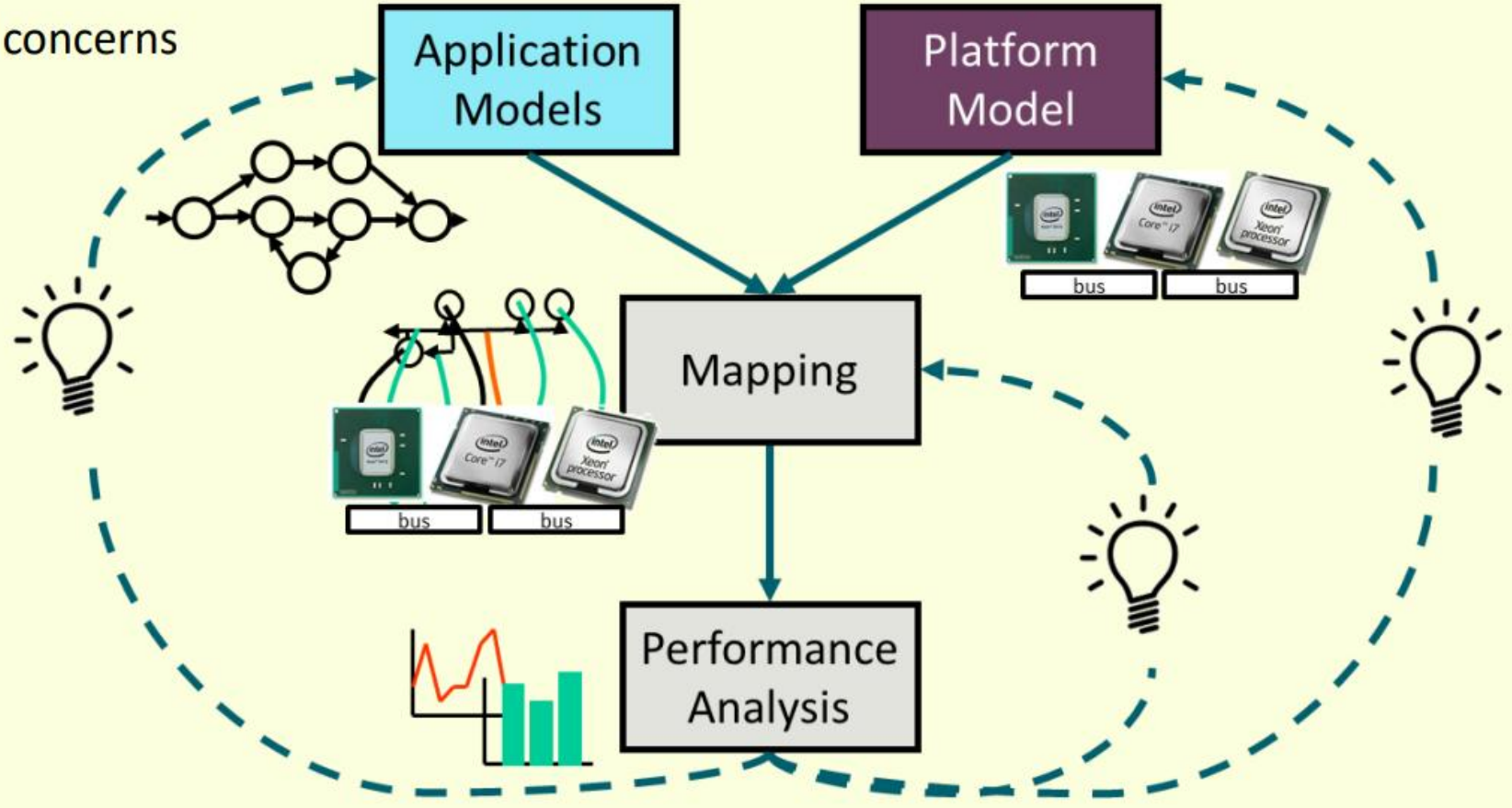
- application: behavior: use cases, activities
- platform: structure: logical and physical: blocks
- mapping of application to platform: relation between form and function

advantages:

- facilitates re-use
- relative independent development of behavior and structure
- provides an efficient way of generating multiple designs for trade off studies

Y-chart

- separation of concerns
- model reuse



xCPS – new system

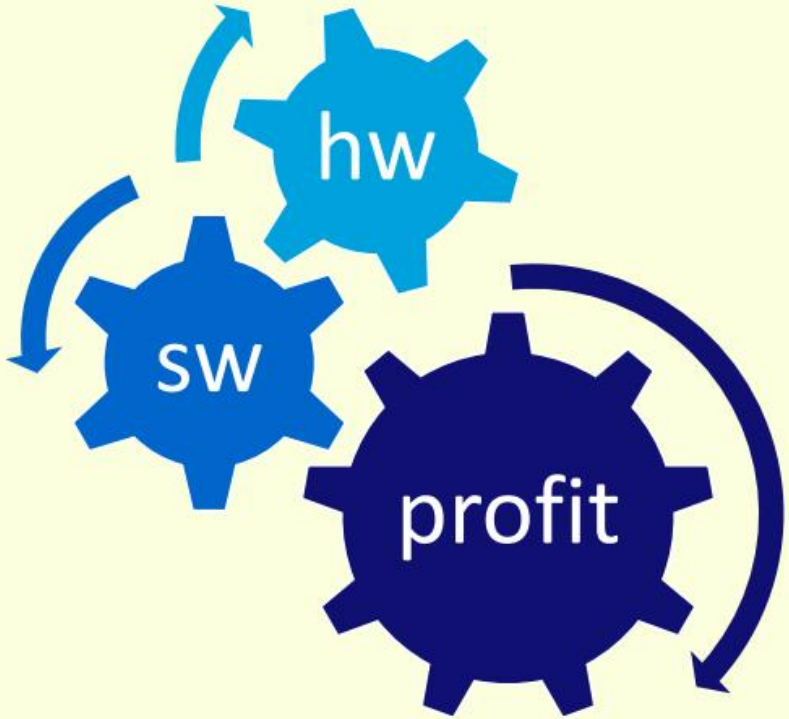
variation points

- hardware
  - slow, normal or fast gantry arm(s)
  - slow, normal or fast index table
  - slow, normal or fast belts
- software
  - piece logistics

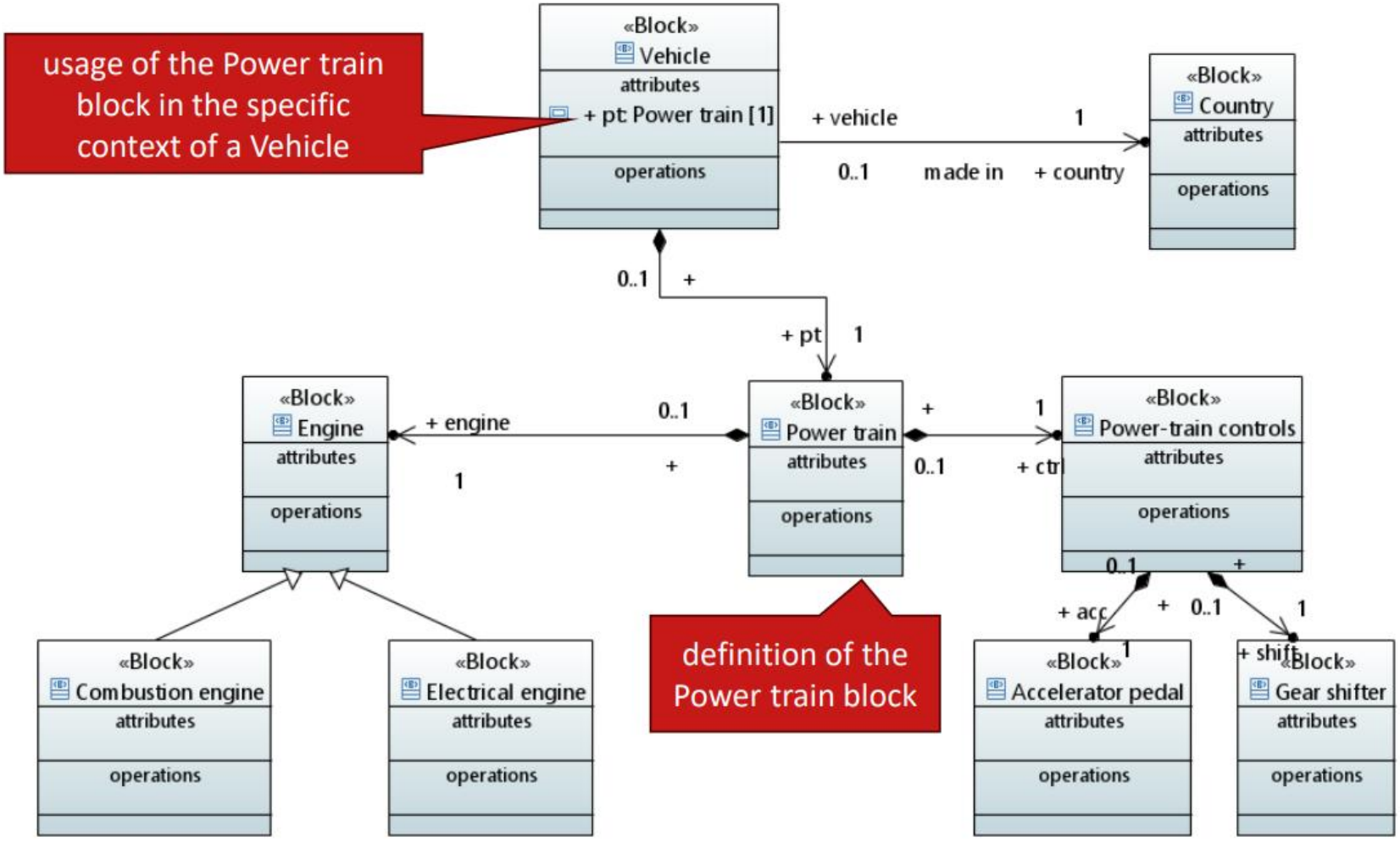
variation impacts

- batch makespan
- bill-of-material
- engineering cost
- time-to-market
- risk

…and all this impacts the profit



SysML – functional allocation – definition vs usage



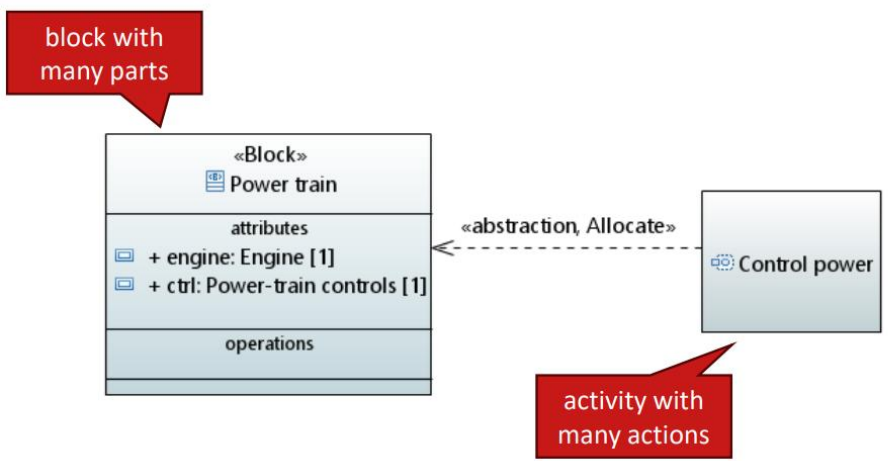
SysML – functional allocation – definition vs usage

allocation of definition: activity to block

- when the allocation is intended to apply to all usages
- can result in over-allocation, i.e., more actions allocated to a part than necessary

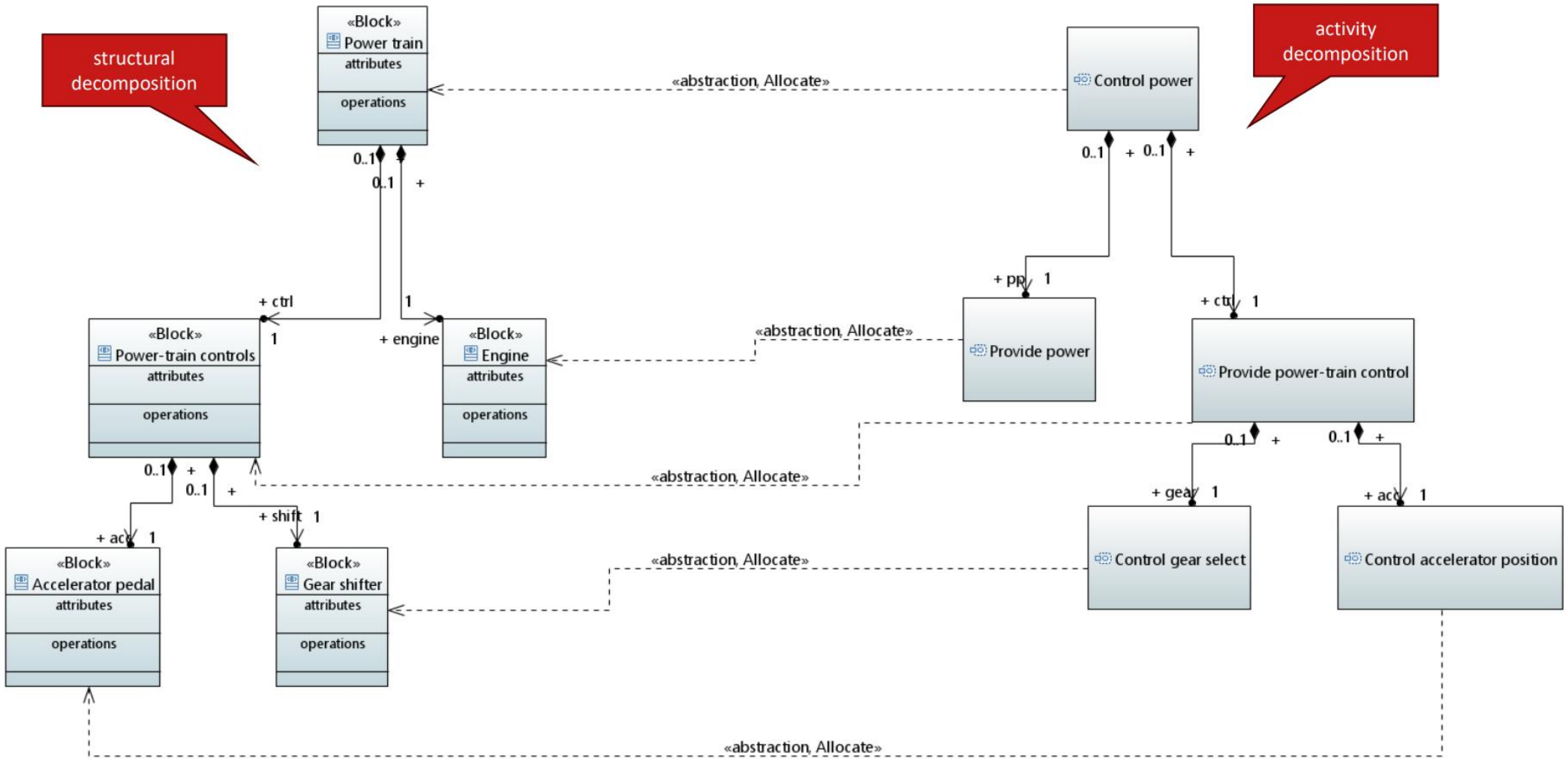
allocation of usage: action to part

- when the allocation is not intended to be re-used
- possible redundancy or inconsistency because parts/actions can be used in multiple places



SysML – functional allocation – definition vs usage

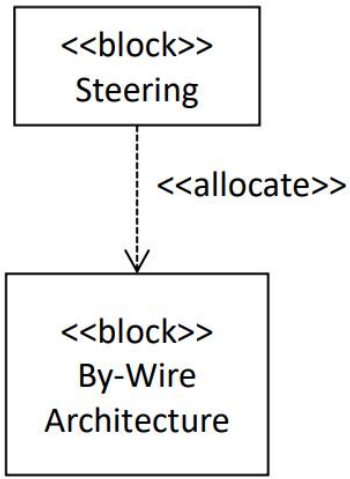
- start with allocation of usage;
- examine each of the uses, then consider allocation of definition
- allocation of definition requires blocks/activities to be specialized or decomposed to the point where the allocation of definition is unique, and over-allocation is avoided


structural decomposition
activity decomposition
«Block»
Power train
attributes
operations
«abstraction, Allocate»
Control power
0..1
+
0..1
+
+ ctrl
1
+ engine
1
«Block»
Power-train controls
attributes
operations
«Block»
Engine
attributes
operations
«abstraction, Allocate»
+ pp
1
Provide power
+ ctrl
1
Provide power-train control
«abstraction, Allocate»
0..1
+
0..1
+
+ shift
1
+ gear
1
+ acc
1
«Block»
Accelerator pedal
attributes
operations
«Block»
Gear shifter
attributes
operations
«abstraction, Allocate»
Control gear select
Control accelerator position
«abstraction, Allocate»

SysML – structural allocation

multiple logical architectures to model the what

- physical parts
- electrical / power supply architecture
- network architecture
- software architecture
- etc.



in the end each logical component will be implemented by a physical part: the how